Building Safe Multi-Agent Applications by Formalising the Deployment Environment

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

GUIRAUD Yves
Laboratoire Informatique, Robotique et Micro Electronique- UMR 5506 - 161, rue Ada 34090 Montpellier cedex 5
34090 Montpellier cedex 5, France
guiraud@math.univ-montp2.fr

MICHEL Fabien
Laboratoire Informatique, Robotique et Micro Electronique- UMR 5506 - 161, rue Ada 34090 Montpellier cedex 5
34090 Montpellier cedex 5, France
fmichel@lirmm.fr

ABSTRACT

This paper presents the MIC* algebraic structure modelling an environment, called the **deployment environment**, where autonomous, interacting and mobile entities evolve. The underlying idea of this work is that the system coherency rules assumptions are guaranteed by the structure of the deployment environment itself and no assumption is made on of the internal behaviours of agents. Thus, every agent has to follow the rules of the deployment environment to being able to interact with others. So, any misuse will be transparently rejected by the structure and other agents are not affected by bogus actions.

1. INTRODUCTION

Nowadays, agent oriented technology is an attractive alternative to build complex distributed applications. Distributing the control among local entities and clearly defining their interactions is a logical abstraction that helps to develop efficient complex distributed systems. Within this framework, building safe software is a real challenge. Indeed, the environment surrounding software system, namely the *deployment environment*, becomes more complex and the properties of this environment cannot be ignored anymore during the engineering process. In fact, as mentioned in [3] ignoring the deployment environment properties may lead to dysfunctions when the applicative system is deployed and these dysfunctions may not be explainable when the system is isolated from its environment. For instance, a complex software system that operates correctly in a fully controlled and predictable environment such as local area networks may not achieve its design goals when deployed on a globally uncontrolled environment such as Internet due to unreliability and latency of the interaction mediums. This shows clearly the correlation between the deployment environment and the applicative system. To solve this problem, the first approach was to develop the software system to a specific deployment environment. This solution is difficult because the properties of the deployment environments are continuously evolving and software designers cannot totally control them. For instance, in the previous example, once the system adapted to the Internet environment, how will it react when deployed in a ubiquitous environment?
The second approach, that is developed in this paper, is to identify the deployment environment as an explicit structure and to study its properties at the design level. Afterward, the system's entities are deployed on this particular environment.

AAMAS'03 Safe Agent Workshop 2003, Melbourne, Australia

1.1 Motivations

This paper argues that understanding and explicitly representing the deployment environment where computational entities evolve is a crucial issue especially for dynamic and unpredictable environments. Therefore, an algebraic model, named {Movement, Interaction, Calculus}* (or MIC* for short), has been developed. MIC* is an abstract model where autonomous, interacting and mobile entities evolve. It should not to be considered as a formal model of a mobile calculus such as Ambient [3], PI calculus [10] or Join calculus [8]. In fact, MIC* suggests a separation between the environment surrounding the calculating entities and the internal calculus of these entities. MIC* studies the environmental properties, while calculus models study the properties of the calculus (algorithm). The motivations of our approach can be summarised as following:

Rigorous description: complex systems use concepts that are overloaded and shared between several research fields. MIC* does not give universal definitions, but at least characterises formally, in the scope of its study, some fundamental concepts such as interaction, movement and observable computation.

Implementation ready model: The MIC* algebraic structure is implemented using simple computational structures that satisfy the constraints imposed by the algebraic objects. Consequently, it is possible to link the concrete computational system that is executed to its corresponding MIC* formal term and obviously it is possible to implement any MIC* formal description in concrete software structure.

For experimental purposes, this paper presents the MIC* and its applications in the scope of ubiquitous software systems. In fact,

ubiquitous systems, that are enabled by ad hoc networking technologies, seem to be the most complex deployment environments that are currently challenging software engineering community and where classical pure engineering approaches, such as object oriented paradigm, fail to give a satisfactory answer. Indeed, the interaction schema among the entities is severally modified by environmental properties and heterogeneous agents may act together spontaneously just in time and just in space. Besides, ubiquitous environments exhibits a *composition* property that was previously unknown in classical software environments. For instance, different systems can be merged spontaneously when joined spatially.

2. BACKGROUNDS

This section presents some research fields that are addressing the problem of controlling the influences of the deployment environment on the applicative system.

2.0.1 Multi-Agent Systems

Multi-agent systems (MAS) consider a computational system as a coherent aggregation of software entities, named *agents*. An agent is an autonomous entity that achieves its local goals by interacting with other agents and its environment [20]. MAS works have been focused on two major directions: studying the internal state of the agents and studying the interactions among the agents. Works such as [4, 18, 16, 9] and the BDI agent architecture are examples of the first tendency. The second direction of research has been illustrated by works on high level interaction languages (such as FIPA ACL [7] and KQML [5]), coordination protocols such as FIPA protocols [7] and the works on the organisational aspects of MAS such as [6, 21]. MAS community has contributed in understanding how to build complex systems involving the collaboration of several distributed entities. However, few works tackle the general study of the environments where the agents are deployed. This point was considered as an implementation problem and the agents' deployment environment was considered as a middleware that offers low level services such as networking and monitoring services. This paper identifies the deployment environment as an active entity within the MAS framework and describes a generic abstract structure of such an environment.

2.0.2 Mobile Computing

Mobile computing studies computational systems, where software components can change execution environment during their life cycle [17]. Similarly to MAS, mobile code components interact[1] together to achieve some specific goals. Mobile code community has already identified the central role played by the *coordination media* to perform *controlled* and *safe* components' interactions [1, 2]. Thus, several coordination media models were proposed such as Lime [14], Tuscon [13] and MARS [2]. The coordination media can be defined as an explicit entity, defined outside the applicative system that performs the interactions between entities. Moreover, it may actively influence the interactions between components and consequently the functionalities of the global system. We propose to generalise this concept as the deployment environment, which achieves not only the interactions between the system's components, but defines also their movement laws and the "acceptable" observations of their computation.

2.1 Formal Mobile Calculus

Formal models of calculus describe formal computational languages. The Lambda calculus [12] is probably the most known and studied formal computational language. Unfortunately, it can expresse just sequential and static algorithms. This leads to the development of several calculus models such as π-calculus [11], Ambient [3] and the Join calculus [8], that handle modern computing concepts such as mobility, location and distribution. MIC* adopts a different view by clearly separating the calculus from its environment. Consequently, the mobility, interaction and the observations of the entities' computations are defined and studied at the environmental level.

3. INFORMAL EXAMPLE

In order to introduce the MIC* formal structure, this section extracts the main concepts starting from a simple ubiquitous application scenario.

3.1 Ubiquitous Electronic Chat Scenario

The ubiquitous electronic chat application emulates verbal conversations between several humans about some specific topics. This kind of applications has already met a success in the Internet context. For ubiquitous environment, the user is no longer connected permanently to a central network, but owns a small device equipped with some ad hoc networking capabilities. Thus, when several users are spatially joined, for instance in a metro station, they can converse together. The general description of the application can be summarised as following: *(i)* each user *participates* in one or several *discussions*; *(ii)* the interaction between the users are conducted by explicitly sending *messages*.

3.2 Interaction Objects

The first reflection concerns the interactions among agents. These interactions are materialised by concrete objects that we identify as *interaction objects*. Interaction objects are structured objects. For instance, they can be composed in simultaneous interactions. Moreover, a special empty interaction object (the zero 0) can be abstractly identified to express 'no interaction'. In the presented scenario, messages represent the interaction objects and receiving simultaneous messages is viewed as receiving a *sum* ($\sum o$) of interaction objects.

3.3 Interaction Spaces

The interaction spaces are abstract locations where several entities interact by exchanging explicitly interaction objects. An interaction space is an active entity that rules the interactions among agents and may alter the exchanged interaction objects. In the ubiquitous chat scenario, each topic is represented by an interaction space, where several *human agents* can exchange messages. To illustrate the active nature of the interaction space, it is easy to imagine some specific topics that determine the participation rules or defines certain messages acceptance policy. Hence, when a message violates the policy of the topic it is simply ignored by the interaction space (reduction to zero); and as opposed to most of current MAS implementations, the interaction actually does not happen[2]. Reduction to zero may appear as a radical alteration of the interaction objects by the interaction space. A more elaborated example can be sketched: for instance, checking and correcting the spelling of messages. Concerning the mobility over the interaction spaces, it is easy to encode the agents' desires to participate in certain topics

[1] In the scope of this paper, we are interested in mobile components that move in order to interact with other components, which excludes load balancing motivated code mobility.

[2] The inboxes of the agents are structurally not changed: $old_{inbox} + 0 = old_{inbox}$. We consider that changing the structure of the inbox is an interaction even if no reaction is observed

as a logical movement inside these interaction spaces. Naturally, an agent can be present in several interaction spaces. This property defines its logical ubiquity.

3.4 Computational Entities or Agents

Agents perceive and react to external interaction objects by a local computation and the emission of other interaction objects in the interaction spaces. These reactions are considered as *attempts* to influence the universe (others). In fact, the reactions are materialised by explicit and discrete interaction objects that are *fully controlled* by the local laws of the interaction space.

3.5 Ubiquity Levels

In the presented scenario, two levels of ubiquity are identified: *physical ubiquity* and *logical ubiquity*. Physical ubiquity can be viewed as the ability to maintain the computational structures of a system everywhere. For instance, when a group of users take together the same metro wagon: the system computational structures are still coherent and independent from the wagon mobility. Logical ubiquity is defined as the ability of an entity to interact coherently and simultaneously as a whole in several interaction spaces. For example, a user sends messages to several topics reacting as a whole to previously received messages.

4. {MOVEMENT, INTERACTION, COMPUTATION}*

Due to space constraints this section presents semi-formally and briefly some aspect of the {Movement, Interaction, Computation}* structure. The algebraic theoretical definitions are omitted.

4.1 MIC* Matrices

In order to present easily the formal structure, a more intuitive view of the manipulated algebraic objects was designed. In fact, matrix representations are familiar to computer scientists and give spatial representation better than complex linear formulas. To present the matrix view, the reader should assume the following minimal definitions:

- $(\mathcal{O}, +)$ represents the commutative group of interaction objects. This means that interaction objects can be composed commutatively by the $+$ law, and that the empty interaction object exists ($0 \in \mathcal{O}$). Furthermore, each interaction object x has an opposite $(-x)$ and $x + (-x) = 0$;

- $\mathcal{A}$ and $\mathcal{S}$ represent respectively the sets of agents and interaction spaces. $\mathcal{S}$ contains a special element: $1 \in \mathcal{S}$ representing the universe as a global interaction space. Moreover, this special element has the following features: *(i)* no interaction between the entities is possible; *(ii)* all the interaction objects can move inside or outside this interaction spaces without restriction.

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 can be proved using the Turing machine model, no further assumptions are made in MIC* about this element.

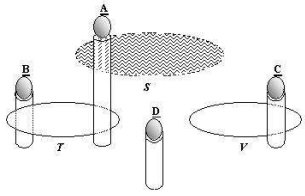

Figure 1: *Agents in an environment.*

	1	S	T	V
A	a	b	c	0
B	d	0	e	0
C	f	0	0	g
D	h	0	0	0

Table 1: Outboxes matrix of figure 1 environment.

For instance, the outboxes matrix presented in table 1 models the situation of the figure 1. When an agent is present in an interaction space, its corresponding interaction object or representation is different from zero. Similarly, a zero represents the fact that an agent is not present in the interaction space.

4.2 Environmental Composition

MIC* terms model naturally ubiquitous environments. In fact, the union or split of computational environments are simply represented as an addition $+$ and a subtraction $-$ defined between the matrices. For instance, let consider two environments e_1 and e_2 where the outboxes matrices are defined as following: $e_1^{outbox} =$

	S_j
A_i	$o_{i,j}$

and $e_2^{outbox} =$

	S_j
$A_{i'}$	$o_{i',j}$

The agents A_i and $A_{i'}$ belong to the same interaction space S_j but are contained in two independent environments e_1 and e_2. Consequently, no interaction is possible between them since their representations are unavailable to calculate the perceptions. Let consider now the union of these environments. $e_3 = e_1 + e_2$: $e_3^{outbox} = e_1^{outbox} + e_2^{outbox} =$

	S_j
A_i	$o_{i,j}$
$A_{i'}$	$o_{i',j}$

The result of this union is a new environment e_3 where the agents $A_{i'}$ and A_i can interact by exchanging their interaction objects. Similarly, any environment can be split into sub environments to model situations where ubiquitous components are disjoint.

4.3 MIC*

The previous section has presented the static objects to describe environmental situations. In this section, we will characterise three main transformations of this static description: the movement, the interaction and the computation (see Figure. 2). A movement is a transformation 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. The interaction

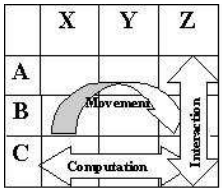

Figure 2: *Movement, Interaction and Computation*

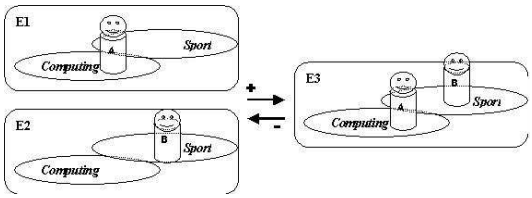

Figure 4: *union and disjunction of environments*

is characterised by a transformation that leaves both outboxes and memories matrices unchanged and transform a row of the inboxes matrix. Thus, interaction is defined as modifying the perceptions of the entities. Finally, an observable computation of an entity transforms its representations in the outboxes matrix and the memories vector.

5. UBIQUITOUS CHAT

5.1 Application Description

Section 3 has introduced ubiquitous chat application emulating human verbal discussions. This demo was implemented using a MIC* prototype written in PYTHON [15] and is fully functional for both LAN and ad hoc networking environments.

5.2 Situation A:

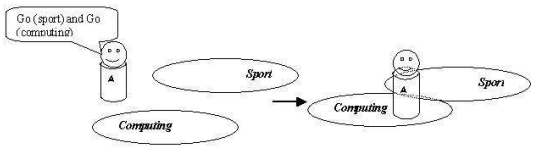

Figure 3: *Agent 'A' moving inside two interaction spaces*

As presented in section 3, each topic is represented by an interaction space. For instance, "sport" and "computing" topics are represented by two interaction spaces (figure 3). When the user selects a chat topic x, the software agent expresses this by sending an interaction object go^x. This interaction object is automatically absorbed by the *correct* interaction space. In fact, the interaction space has a full control of its local movement policy allowing certain interaction objects to enter and refusing the access to others. In the presented scenario, the movement policy of an interaction space x is to absorb all interaction objects go^x and to move outside $-go^x$ interaction objects. The situation expressed in figure 3 can be described formally by the following outboxe matrices:

$$e_0^{outbox} = \begin{array}{|c|c|c|c|} \hline & 1 & sport & computing \\ \hline A & go^{sport} + go^{computing} & 0 & 0 \\ \hline \end{array}$$

that evolves to :

$$\mu(\mu(e_0^{outbox})) = e_1^{outbox} = \begin{array}{|c|c|c|c|} \hline & 1 & sport & computing \\ \hline A & 0 & go^{sport} & go^{computing} \\ \hline \end{array}$$

After these two movements, agent A exists in both interaction spaces: $sport$ and $computation$.

5.3 Situation B:

As illustrated in figure 4, when two environments E_1 and E_2 are joined a new environment E_3 is defined. In this environment, the interaction schema among the entities is modified. For instance, agents A and B are now able to interact since they belong to same interaction space, $sport$, defined in the same environment. On the other side, when the physical network link is disconnected, the environment E_3 is split into E_1 and E_2. This situation is formally described by the following outboxes matrices:

$$\begin{array}{|c|c|c|c|} \hline & 1 & sport & computing \\ \hline A & 0 & go^{sport} & go^{computing} \\ \hline \end{array} + \begin{array}{|c|c|c|c|} \hline & 1 & sport & computing \\ \hline B & 0 & go^{sport} & 0 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|} \hline & 1 & sport & computing \\ \hline A & 0 & go^{sport} & go^{computing} \\ B & 0 & go^{sport} & 0 \\ \hline \end{array}$$

5.4 Situation C:

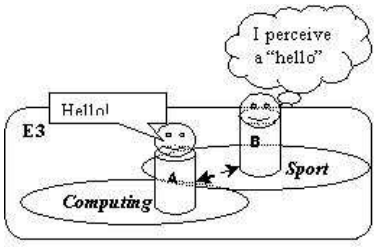

Figure 5: *Interaction among agents*

Computation is an internal process of an agent that modifies its memory (Turing model [19]). Consequently, an agent does not modify the state the surrounding universe directly, but by sending some interaction objects. For instance, when a human agent computes internally what he should write as message, the observation of this process is the written message (interaction object) that is yielded in the interaction space. The surrounding entities receive this interaction object through the interaction space (see figure 5). For instance, when agent A writes a $hello$ message, the outboxes matrix is changed as following:

$$\begin{array}{|c|c|c|c|} \hline & 1 & sport & computing \\ \hline A & 0 & go^{sport} & go^{computing} \\ B & 0 & go^{sport} & 0 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|} \hline & 1 & sport & computing \\ \hline A & 0 & hello & hello \\ B & 0 & go^{sport} & 0 \\ \hline \end{array}$$

The following inboxes matrices describe interaction among agents A and B:

$$\begin{array}{|c|c|c|c|} \hline & 1 & sport & computing \\ \hline A & 0 & 0 & 0 \\ B & 0 & 0 & 0 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|c|} \hline & 1 & sport & computing \\ \hline A & 0 & hello & hello \\ B & 0 & hello & 0 \\ \hline \end{array}$$

Both agents A and B receive the $hello$ message that was emitted in the outboxes matrix. Therefore, they can consider this interaction for their future computations.

6. CONCLUSION

This paper has presented, semi-formally, the MIC* algebraic structure modelling combinable environments where mobile, autonomous and interacting entities evolve. Complex environmental evolutions are described as the composition of three atomic evolutions: the movement, the interaction and the computation. These evolutions are formally characterised using the MIC* matrices.

The ubiquitous chat experiment has demonstrated how this structure is integrated into a simple (human) agents system that interact together when specially joined. For a particular agent based system, the main goal of the design process is to define particular elementary evolutions of movement, interactions and acceptable observation of entities' computation, which yields a particular deployment environment where applicative agents are deployed. These agents may be locally (trusted) defined or encounter dynamically (untrusted) during the system life cycle. To deal with untrusted environments, the system coherency rules and assumptions are guaranteed by the structure of the deployment environment itself and no assumption is made on of the internal behaviours of agents. For instance, it is quite easy to conceive an interaction space that checks the coherency of a particular interaction protocol. Hence, the coherency of the protocol, as an engineering design assumption, is guaranteed by the interaction space, and all agents belonging to this interaction space have to follow it. Any misuse will be transparently rejected by the structure of the interaction space and other agents are not affected by this bogus action.

The next step of our work is to generate the deployment environment automatically starting from system description using organisational and dependency theory. Hence, each dependency between social agents yields an interaction space where agents interact to resolve the dependency following a specific interaction protocol. The interaction space guarantees all the social norms on interaction, mobility and other engineering design assumptions. The main goal is to conceive ubiquitous systems as virtual mobile communities that interact when joined spatially (as in human migrant communities). Each community has its own internal organisation model and interact with other communities through social and organisational interfaces.

7. REFERENCES

[1] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. Coordination in mobile agent applications. Technical Report DSI-97-24, Dipartimento di Scienze dell Ingegneria Universitá di Modena, 1997.

[2] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. Reactive tuple spaces for mobile agent coordination. *Lecture Notes in Computer Science*, 1477, 1998.

[3] Luca Cardelli. Abstraction for mobile computation. *Secure internet programming: security issues for mobile and distributed objects, LNCS 1603*, 1999.

[4] P.R. Cohen and H.J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3), 1990.

[5] DARPA. Specifiaction of the kqml agent-communication language. Technical report, DARPA, 1993.

[6] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems, 1998.

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

[8] Cedric Fournet. *Le join-calcul: un calcul pour la programmation repartie et mobile*. PhD thesis, Ecole Polytechnique, 1998.

[9] N.R. Jennings. On being responsible. *Decentralised AI*, 3, 1992.

[10] Robin Milner. Communication and mobile systems: the pi-calculus. Cambridge University Press, 2000.

[11] Robin Milner, Joachim Parrow, and David Walker. A calculus for mobile processes, parts 1 and 2. *Information and computation*, 1992.

[12] James Hiram Morris. λ-calculus model of programming language, 1968. MIT.

[13] A. Omicini and F. Zambonelli. The tucson coordination model for mobile information agents. *1st Workshop on Innovative Internet Information Systems*, 1998.

[14] Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. Lime: Linda meets mobility. In *International Conference on Software Engineering*, pages 368–377, 1999.

[15] Python. Python web resources. web, March 2002. http://www.python.org.

[16] A.S. Rao and M.P. Georgeff. Modelling rational agents within a bdi-architecture. In *2nd Inter. Conference on Principles of knowledge Representation and Reasoning*, 1991.

[17] G.-C. Roman, G. P. Picco, and A. L. Murphy. Software engineering for mobility: A roadmap. *The Future of Software Engineerin,*, pages 241–258, 2000.

[18] M. Singh and N. Asher. Towards a formal theory of intentions. *Logics in AI*, LNAI:478:472–486, 1990.

[19] Alan Turing. On computable numbers, with an application to the entscheidungsproblem. In *Proceedings of the London Mathematical Soceity*, number 42 in 2, 1936.

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

[21] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.