MIC*: An Agent Formal Environment

Abdelkader GOUAICH[1], *Yves GUIRAUD*[1,2], *Fabien MICHEL*[1]

[1]LIRMM, Montpellier
[2] Laboratoire GTA, Université Montpellier2, Montpellier
{gouaich,yguiraud,fmichel}@lirmm.fr

Abstract

Ubiquitous systems can be considered as *good* challenges and case studies for the reliable development of open, cooperative and mobile systems in a globally uncontrolled environment. Furthermore, the complexity of these systems could constitute a real barrier to classical engineering approaches and validate the foundations of agent based computing. In this paper, we have abstracted a composable multi-agent system environment in an algebraic structure where mobile, interacting and autonomous entities evolve. We present also, a simple ubiquitous application developed according to the abstract model principles.

keywords: agent formal deployment environment, agent-based software system, ubiquitous software system.

1. INTRODUCTION

Network technologies are offering the opportunity to interconnect several heterogeneous software systems anytime and anywhere. Within the context of these systems, commonly identified as *ubiquitous systems*, the surrounding environment, which we identify as the *deployment environment*, influences the interaction scheme of the entities and thus affects considerably the characteristics and the functionalities of the global system. This phenomenon has already been identified by L. Cardelli in [Car99], where he argues that the design and the realisation of a complex software system should handle the properties of the targeted deployment environment: LAN, WAN or mobile environment. This leads to the clear identification of a new entity, which was previously ignored in software system engineering: *the deployment environment* [GG02a]. F. Zambonelli and H. Parunak in [ZP02] have also mentioned that intergrading the deployment environment in software engineering process is a crucial issue especially for complex and dynamical systems. Furthermore, they have described some general properties of future deployment environments such as:

- situatedness: software components are executed in the context of an environment;
- openness: software systems are decentralised and dynamically change their structure;
- locality in control: the components of software systems are autonomous and proactive computational entities;
- locality in interactions: despite living in a fully connected world, software components interact accordingly to local patterns.

Besides, ubiquitous environments exhibit a *composability property* allowing them to be joined or separated. For instance, when some physical (or logical) conditions are satisfied, such as distance or signal intensity, a new environment is created by the composition of sub environments. The entities of this new composed environment may behave differently since they can interact with entities that were previously unavailable. Similarly, when these environmental conditions are unsatisfied, the global system is split into several independent sub systems.

1.1. Motivations

In this paper, we argue that understanding and explicitly representing the deployment environment where computational entities evolve is a crucial issue especially for dynamic and unpredictable environments like ubiquitous environments. Therefore, we have defined an algebraic model named {Movement, Interaction, Computation}* (or MIC* for short). MIC* is an abstract model where autonomous, interacting and mobile entities evolve. It has not to be considered as a formal model of a mobile calculus such as Ambient [Car99], PI calculus [Mil00] and the Join calculus [Fou98]. 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). Finally, the motivations of our approach can be summarised as following:

Rigorous description: ubiquitous and more generally 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 system.

1.2. Paper outlines

This paper is organised as following: section 2 presents some related fields that deal with similar problems encountered in ubiquitous environments; section 3 introduces informally a simple ubiquitous application scenario; section 4 introduces semi formally the MIC* structure; section 5 presents the implementation of the application; and finally, section 6 concludes and gives future perspectives.

2. Related fields

The importance of the deployment environment was already identified in both multi-agents systems (MAS) and mobile code communities. This section presents briefly these research fields.

2.1. 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 [WJ95]. Since the components of ubiquitous systems behave autonomously and interact with other entities, they can be considered naturally as agents and may benefit from several MAS works on high level interaction languages (such as FIPA ACL [FIP96a] and KQML [DAR93]) and coordination protocols (for instance FIPA Protocols [FIP96b]) to allow interoperability between heterogeneous systems. On the other side, MAS community has some difficulties to show what are the real contributions and added values of its approach. In fact, for simple systems, the object-oriented approach is a well satisfying solution. However, dynamic, mobile and complex environments such as ubiquitous systems could constitute a breaking point between traditional paradigms and the MAS approaches for complex systems software engineering.

2.2. Mobile computing

Mobile computing studies computational systems, where software components can change execution environment during their life cycle [RPM00]. 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 [CLZ97, CLZ98]. Thus, several coordination media models were proposed such as Lime [PMR99], Tuscon [OZ98] and MARS [CLZ98]. 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.

2.3. Mobile formal calculus

Calculus formal models describe a formal programming language. The λ-calculus [Mor68] is probably one of the most known of these models. Unfortunately, λ-calculus describes just sequential and static algorithms. In order to describe some modern computing concepts such as distribution, mobility and security, others formal calculus were developed. Among these models, one can find the π-calculus [MPW92] (and its derived models for distribution handling Dπ [AG98] and security management Sπ[MH98]); Ambient [Car99] and the Join calculus [Fou98]. MIC* is an alternative approach that defines mobility, distribution and interaction concepts at the deployment environment level and not at the calculus or algorithmic 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* ($\sum o_i$) 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 sets some 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 contrary 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 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.

[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

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 limitation, this section presents semi-formally and briefly some aspect of the $\{$Movement, Interaction, Computation$\}^*$ structure. A full mathematical presentation is given in [GG02b].

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.

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.

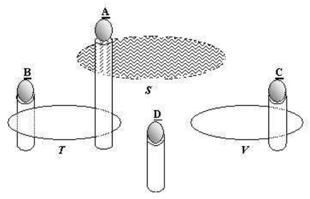

Figure 1: *Agents in an environment.*

	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.

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: $e_1^{outbox} = \begin{array}{c|c} & S_j \\ \hline A_i & o_{i,j} \end{array}$ and $e_2^{outbox} = \begin{array}{c|c} & S_j \\ \hline A_{i'} & o_{i',j} \end{array}$

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. $e_3 = e_1 + e_2$: $e_3^{outbox} = e_1^{outbox} + e_2^{outbox} = \begin{array}{c|c} & S_j \\ \hline A_i & o_{i,j} \\ A_{i'} & o_{i',j} \end{array}$

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*

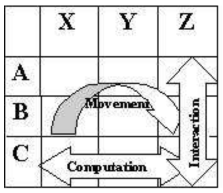

Figure 2: *Movement, Interaction and Computation*

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 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.

4.4. Limitations

MIC* is a *descriptive* formal approach, which means that it gives some abstract elements to model concrete systems and defines rigorously some concepts such as *movement*,*interaction* and *observable computation*. Until now, no general formal tools are given to prove or to predict some properties of the environmental model.

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[Pyt02] and is fully functional for both LAN and ad hoc networking environments.

5.2. Situation A:

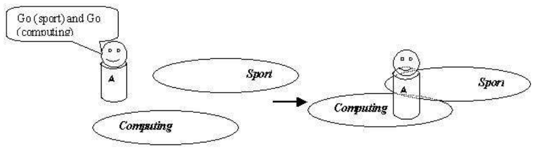

Figure 3: *Agent 'A' moving inside two interaction spaces*

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 express 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 outboxes matrices:

$$e_0^{outbox} =$$

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

that evolves to :

$$\mu(\mu(e_0^{outbox})) = e_1^{outbox} =$$

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

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

5.3. Situation B:

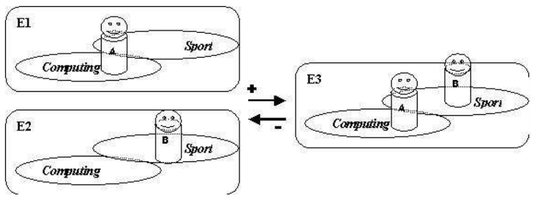

Figure 4: *union and disjunction of environments*

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:

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

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

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

5.4. Situation C:

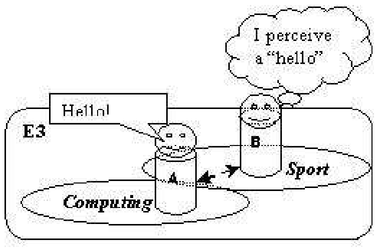

Figure 5: *Interaction among agents*

Computation is an internal process of an agent that modifies its memory (Turing model [Tur36]). 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:

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

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

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

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

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

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

In this paper, we have presented the MIC* algebraic structure modelling combinable environments where mobile, autonomous and interacting entities evolves. The next step of our work is to generate specific environments and interaction spaces starting from the engineering specification of a system. Following the MIC* approach, it would be possible to guarantee these specifications in unpredictable and dynamical environments such as ubiquitous systems.

7. References

[AG98] M. Abadi and A.D. Gordan. A calculus for cryptographic protocols: The spi calculus. Information and Computation, 1998.

[Car99] Luca Cardelli. Abstraction for mobile computation. *Secure internet programming: security issues for mobile and distributed objects, LNCS 1603*, 1999.

[CLZ97] 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.

[CLZ98] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. Reactive tuple spaces for mobile agent coordination. *Lecture Notes in Computer Science*, 1477, 1998.

[DAR93] DARPA. Specifiaction of the kqml agent-communication language. Technical report, DARPA, 1993.

[FIP96a] FIPA. Fipa agent communication language, www.fipa.org/, 1996.

[FIP96b] FIPA. Foundation of intelligent and physical agents, 1996. http://www.fipa.org.

[Fou98] Cedric Fournet. *Le join-calcul: un calcul pour la programmation repartie et mobile*. PhD thesis, Ecole Polytechnique, 1998.

[GG02a] A. Gouaich and Y. Guiraud. (movement, interaction, calculus)* : an algebraic environment for distributed and mobile calculus. In *ICAIS*. Deakin University, February 2002.

[GG02b] Abdelkader Gouaich and Yves Guiraud. Mic* algebraic structure. Technical report, LIRMM, 2002. http://www.lirmm.fr/˜gouaich/research.html.

[MH98] James Riely Mathew Hennessy. A typed language for distributed mobile processes. In Proc. Of POPL, ACM Press, 1998.

[Mil00] Robin Milner. Communication and mobile systems: the pi-calculus. Cambridge University Press, 2000.

[Mor68] James Hiram Morris. λ-calculus model of programming language, 1968. MIT.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus for mobile processes, parts 1 and 2. *Information and computation*, 1992.

[OZ98] A. Omicini and F. Zambonelli. The tucson coordination model for mobile information agents. *1st Workshop on Innovative Internet Information Systems*, 1998.

[PMR99] Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. Lime: Linda meets mobility. In *International Conference on Software Engineering*, pages 368–377, 1999.

[Pyt02] Python. Python web resources. web, March 2002. http://www.python.org.

[RPM00] 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.

[Tur36] Alan Turing. On computable numbers, with an application to the entscheidungsproblem. In *Proceedings of the London Mathematical Soceity*, number 42 in 2, 1936.

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

[ZP02] Franco Zambonelli and H. Van Dyke Parunak. Signs of a revolution in computer science and software engineering. In *AOSE 02, AAMAS 2002*, Bologna, 2002.