Distributed Ubiquitous Software Services

GOUAICH Abdelkader
Laboratoire Informatique, Robotique et Micro Electronic- UMR 5506
- Montpellier - France
gouaich@lirmm.fr

Abstract

This paper presents both theoretical and experimental results enabling the distributed implementation of ubiquitous software services.

1 Introduction

The availability of new generation of computing and communication technologies people are offered the opportunity to use software services anywhere and anytime. These services are known as *ubiquitous software services*. The ubiquity of a software service is defined as its ability to provide its functionalities to other components (clients) coherently and continuously everywhere within a defined space. By contrast to most current approaches that adopt a centralised vision by assuming that the distributed software system composed by the ubiquitous service and its clients are able to interact everywhere, this paper presents a different approach enabling a globally coherent ubiquitous service implemented by several distributed and disconnected entities. These *service instances* use only local communication technologies.

1.1 MIC* agent deployment environment model

In order to understand the environment surrounding a multi-agents system an algebraic model, {Movement, Interaction, Computation}*, has been introduced in [3]. It defines an abstract structure where autonomous, mobile and interacting entities may be deployed. Within this framework, agents interact by exchanging *interaction objects*, found in a set represented as $\mathcal{O}$, through *interaction spaces*. The mobility of the agents is described as the mobility of interaction objects over interaction spaces. MIC* describes the dynamics of the environmental structure as a composition of three elementary evolutions: the movement, the interaction and the computation.

2 Theoretical foundations

In this section some theoretical definitions are given in order to prove that similarity among distributed and disconnected components behave as a ubiquitous entity. A dialogue represents a sequence of interactions. Within the MIC* framework, a dialogue can be represented by a *sequence* of interaction objects generated by the free monoid $\mathcal{O}^*$ that is built using, . (dot), a non-commutative law on interaction objects. The sequence structure is used as an interaction object to encode the *memory* of the conversation. The ubiquity property is defined as an ability to be present everywhere within a defined space [1]. Formally, this definition is expressed as follows:

Definition 2.1. *A computational entity $p \in \mathcal{A}$ is said to be ubiquitous within a defined topology $\delta \subset \Delta$ iff:*

$$\begin{aligned} \forall x \in \delta, \pi(p,x) &= 1 \\ \forall o \in \mathcal{O}^*, \forall x,y \in \delta, \gamma(p_x,o) &= \gamma(p_y,o) \end{aligned}$$

Where Δ represents a space. $\pi : \mathcal{A} \times \Delta \rightarrow \{0,1\}$ is a characteristic function that equals 1 when a process $p \in \mathcal{A}$ is located in $x \in \Delta$ and 0 otherwise. $\gamma : \mathcal{A} \times \mathcal{O}^* \rightarrow \mathcal{O}^*$ is the calculus function presented in MIC*, that maps the perceptions of an agents to its observable emitted interaction objects. The notation p_x is used to identify the process p located at coordinate x of the space. The first assertion of definition 2.1 expresses the fact that the entity has to be located everywhere in δ. The second assessment expresses that from an external point of view, the entity behaves as a single process.
The mapping relation defines where the software processes are actually located in the space. This is defined formally as follows:

Definition 2.2. *A mapping, m, is a relation defined on $\mathcal{A} \times \Delta$ that links processes to space coordinates.*

An agent process $p \in \mathcal{A}$ is said to be similar to another agent process $q \in \mathcal{A}$ when both are observed similarly by an external observer. This definition is formalised as follows:

Definition 2.3. *Similarity $\sigma : \mathcal{A} \times \mathcal{A}$ relation is defined as follows: $\sigma(p,q) \Leftrightarrow \forall o \in \mathcal{O}^*, \gamma(p,o) = \gamma(q,o)$*

Obviously, σ is an equivalence relation on $\mathcal{A} \times \mathcal{A}$. Thus, σ defines an equivalence class of an element p as follows $\hat{p} = \{q \in \mathcal{A} : \sigma(p,q)\}$. The notation, $\hat{p}$, represents then the set of processes that are similar to p.

Definition 2.4. *The covered topology $\delta_{\hat{p},m}$ of an equivalence class $\hat{p}$ according to the mapping m is defined as follows: $\delta_{\hat{p},m} = \bigcup_{p \in \hat{p}} \{x \in \Delta | (p,x) \in m\}$*

$\delta_{\hat{p},m}$ represents the space covered by all the similar processes according to a particular mapping m.

Proposition 2.5. *Let Δ be a space, $\mathcal{A}$ an agent set, m a $(\mathcal{A} \times \Delta)$-mapping. If $\hat{p}$ is σ-equivalence class, then $\hat{p}$ is a ubiquitous entity on $\delta_{\hat{p},m}$.*

The first assessment of definition 2.1 is verified since all processes belonging to $\hat{p}$ belong to $\delta_{\hat{p},m}$ according to definition 2.4. Concerning the second part of the ubiquity definition, if $\hat{p}$ is not a ubiquitous entity, this means that:
$\neg ubiquity(\hat{p}, \delta_{\hat{p},m})$
$\Rightarrow \exists o \in \mathcal{O}^*, \exists x, y \in \delta_{\hat{p},m} : \gamma(\hat{p}_x, o) \neq \gamma(\hat{p}_y, o)$
$\Rightarrow \exists p, q \in \hat{p} \wedge (p,x), (q,y) \in m : \gamma(p,o) \neq \gamma(q,o)$
$\Rightarrow \neg\sigma(p,q)$, which contradicts $p, q \in \hat{p}\square$
Proposition 2.5, theoretically proves that similarity between independent and autonomous processes naturally builds a ubiquitous entity spread over the space occupied by all its processes. The next section shows how to implement the similarity relation between autonomous and disconnected processes.

3 Achieving similarity in distributed and disconnected environment

In order to behave coherently, each instance of the ubiquitous service $\hat{p}$ should be able to continue coherently any interaction that was initiated with another σ-equivalent process. To achieve this, ubiquitous service instances have to know about history of ongoing interactions. This interaction memory can be shared among the instances following different mechanisms.

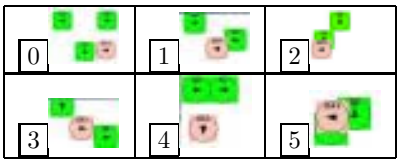

Figure 1. Elementary situations between a user agent (ua), light grey box, and several ubiquitous service instances (s): dark grey boxes

For instance, a common accessible storage space can be used to share information. However, implementing this mechanism is difficult in a disconnected and asynchronous environment. Encapsulating the interaction memory in the interaction itself seems to be an interesting alternative. Hence, any instance of the service is able to rebuild locally the memory of the ongoing dialogue when needed. Having this memory of the dialogue, the service instance is then expected to react similarly to other instances. According to proposition 2.5, this defines a virtual ubiquitous service. Using the dialogue as an interaction object raises also some problems that are summarised as follows:

- 'Remember' a dialogue-level communicative act: The problem appears when the client agent interacts with the ubiquitous service (situation 1 in figure 1) and is suddenly disconnected while expecting a response (situation 0 in figure 1). When reconnected to another instance of the ubiquitous service (situation 1 in figure 1), the newly met service instance has no knowledge about the ongoing dialogue and the client agent is waiting for an answer. This creates a dialogue deadlock. This deadlock is resolved by introducing a neutral communicative act about the conversation. Hence, the client agent asks the service agent to *remember* what was the last interaction stream.

- Ubiquitous service instances synchronisation : Although ubiquitous service instances are distributed and autonomous, some critical actions implying side effects have to be synchronised. For instance, the payment action for a particular service should be performed once by a service instance and the user should not be charged again by other instances. This is an important feature to be guaranteed in order to meet ubiquitous service definition. In fact, if the service's client observes any inconsistency then the ubiquity property presented in definition 2.1 is lost. Consequently, service instances have to coordinate their side-effects

actions.

4 Experiments

In order to correctly simulate ubiquitous environment a three layers architecture has been used:

1. Movement simulator: The movement simulator represents the low-level geometric communication range of the entities and handles their movement in a two dimensional space.

2. MIC* environment: a MIC* term represents the deployment environment of agents. Thus, agents' processes are deployed on MIC* terms that are linked when their geometric representations overlap.

3. Agents: agents' processes react to perceived interaction objects by emitting other interaction objects. Since agents interact only when they are in the same deployment environment, the geometric overlapping becomes a sine qua non condition to agents' interactions.

The experiment scenario simulates a virtual city where both user agents and service agents move and may interact. For this demonstration a single ticket buying service has been implemented in the virtual city. The user agent goal is to buy a ticket from a ticket selling service while moving in the city. Notice that service agents also move which raises the dynamics of the system. The user agent is expected to perceive a single coherent ubiquitous service implemented by several distributed and disconnected entities.

4.0.1 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

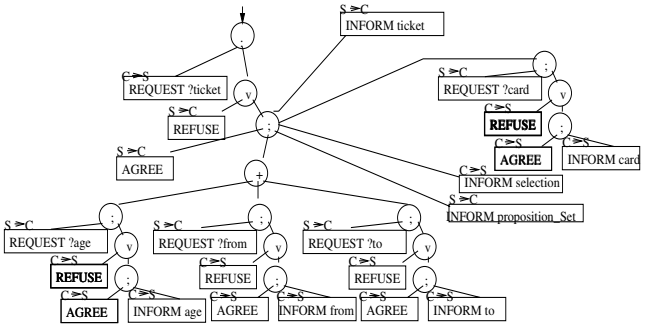

Figure 2. Ticket buying interaction protocol between the user agent (C) and the service (S)

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 [2]. All autonomous agents are executed as concurrent processes that interact with the deployment environment with explicit interaction objects sent via low-level TCP/IP sockets. Despite disconnections due to the mobility of the entities, the user agent succeeds each time in buying a travel ticket. The debug traces show also that several service instances have been involved in the interaction with the user agent. The user agent perceives a single coherent ubiquitous service spread in the surface occupied by its instances. The experimental work presented in this section validates the theoretical results presented in section 2.

5 Conclusion

This paper has defined the ubiquity of a software entity as its presence, as a coherent whole, in a certain spatial topology. The theoretical result has proved that achieving similarity among software processes defines a virtually distributed ubiquitous entity. Using the composition property of the MIC* formal model and defining dialogue structure, the similarity property has been successfully implemented in a highly distributed and disconnected context. The implementation of this property has also raised some problems such as the need of a neutral communicative, 'Remember', act to

restart a freezed dialogue. Consistency of the memory of the dialogue is also an important issue that has not been specifically addressed in this paper and should be handled by future works.

References

[1] *Le Petit Larousse Illustré*. LAROUSSE, 2001.

[2] A. GOUAICH. Interaction conformity in distributed and disconnected multi-agents systems. In *The IEEE/WIC International Conference on Intelligent Agent Technology*, Halifax, Canada., 2003.

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