Ubiquitous computing : vanishing the notion of application

P. Mathieu
Laboratoire d'Informatique
Fondamentale de Lille
Cité Scientifique 59655
Villeneuve d'Ascq Cedex
mathieu@lifl.fr

J.C. Routier
Laboratoire d'Informatique
Fondamentale de Lille
Cité Scientifique 59655
Villeneuve d'Ascq Cedex
routier@lifl.fr

Y. Secq
Laboratoire d'Informatique
Fondamentale de Lille
Cité Scientifique 59655
Villeneuve d'Ascq Cedex
secq@lifl.fr

ABSTRACT

This paper postulates that multi-agent systems are the right choice for the design and implementation of pervasive frameworks. This claim relies on the idea that the notion of application is a non-sense in a pervasive environment because of its highly dynamic characteristics. Instead, we propose the use of an interaction based design, relying on multi-agent concepts like agents incrementally built from skills, role decomposition, reification of interactions and organizations. We will first give a brief overview of available concepts and technologies to support pervasive computing, before introducing our interaction-based design relying on our minimal agent model.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design tools and techniques; I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Design, Languages, Standardization

1. INTRODUCTION

Ubiquitous computing has been an old dream that is probably born when people realized the potential of inter-connected personal computers. Some works were done in the late 80's[2], that show the early interest people had in this idea that computers could be faded within our environment like writings and pictures. Nowadays, several important factors have deeply changed the environment : devices are really surrounding peoples (personal computers, notebooks, personal digital assistants, mobile phones), and networks are omnipresent (internet, Bluetooth, GSM). What is important to notice is that technologies and economical interest are driving today massive developments on both hardware and software. Sadly, these developments are generally *ad hoc*, and can not be reused or extended. These solutions are tailored to suit clients needs and are therefore closer to dedicated applications than to frameworks for pervasive computing. Of course, all these economical and technological factors are important, but they are not sufficient; and to provide some kind of methodology and software infrastructure is necessary.

AAMAS'02:UbiAgents Workshop Bologna, Italy

2. MULTI-AGENTS PARADIGMS

Multi-agent systems (MAS) rely on some key concepts : the notion of agent, the notion of society and finally the notion of organization. We will not give here definitions of these notions, as there are actually no formal definitions available. Instead, we will describe our propositions for these notions.

An agent should be seen as an empty shell that has only two abilities : communication and evolution[3]. The first ability allows us to interact with the agent, while the second one is necessary in order to tailor the agent to its tasks. Then, all agents are equal at birth, and evolve by acquiring new skills. These skills enable them to fulfill the roles they have been assigned in the society they belongs to. This notion of minimal generic agent is important as it defines a system level that has not yet been identified in the MAS field: there are a lot of agent models and agent platforms, but there is not a lot of work to identify a common infrastructure among these propositions. Our thesis is that this notion of minimal generic agent allows us to build any kind of agent model by providing the model as a skill. Developers only have to define the fundamentals of their agent model through skills.

This notion is strongly related to the notion of role : an agent can handle multiple roles, and the behavior of these roles are implemented through skills. Software engineering practices have promoted the notion of modularization. A skill can be seen as a component that accepts and produces some particular kinds of messages. Then, a skill can be described as a set of couples of messages. Each couple represents a valid incoming message and its corresponding answer. A message is defined in an XML language, and is identified by its grammar and its main tag. As a skill can be deployed in several runtime environments, it is important to uncouple its interface from its implementation. Definitions given above describe the interface of a skill : messages accepted as input and their answers. The implementation of a skill will depend of the underlying component model and the runtime environment.

The representation of these informations should be done through a language independent description. The language we have chosen is DAML+OIL [1]. From a technical point of view, DAML+OIL can be thought as a RDF language relying on first order logic to describe its semantic. Thus, DAML+OIL is the ideal language to describe ontologies that can be complex but that remain tractable in terms of inferences that can be done.

Working with one agent is one thing, but it is of course when we have to deal with multiple agents that this approach gives its meaning. An organization is necessary to structure interactions between roles within the system. This notion brings several important features : a default communication path between agents, a mean to logically organize agents, its reification provides a hook to monitor

[1] http://www.daml.org

and improve agents interactions. This notion of organization can be seen as the topology of acquaintances graph. For example, in a hierarchical organization it will look like a tree, while in a group based one it will be a sparse graph. Another important feature of organizations is that they can dynamically evolve to enhance flows of interactions. This dynamicity of the organization ease the burden of the designer and provide more reliability to the system.

3. INTERACTION-BASED DESIGN

The main idea is to reify the notion of interaction to give to designers a global view of the interaction on all aspects : coordination, messages exchanged, skills needed. Thus, an interaction is made of three components : roles that are involved, messages that are exchanged, services that are necessary to handle these messages. Then, an interaction can be seen as an oriented graph where each node represents a state of the conversation, and edges represent messages exchanged between these states. Each state is linked with the service that should handle this part of the conversation, while edges are typed by the nature of messages. To illustrate these notions, let us take the example of a simple e-commerce interaction. The figure 1 illustrates what the developer should see while

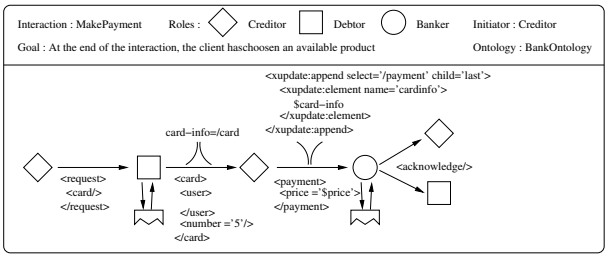

Figure 1: The interaction describing a payment

designing its system : a global view of the coordination and side effects of an interaction. Regardless of the formalism that will be used to specify interactions (enhanced Dooley graphs, Colored Petri Nets[1]), the key idea is to keep informations about coordination and messages handling in one representation.

This representation of the interaction is made of all necessary informations : the name of the interaction, the roles involved in this interaction, initiator(s) role(s), ontology that contains the definition of all messages exchanged, a description of the interaction flow, and skill interfaces.

In the figure 1, we have chosen a to represent the interaction between a *Creditor*, a*Debtor* and a *Banker*. The key point of this description is that all necessary informations are available : roles involved, nature of messages exchanged, skill interfaces necessary to handle messages, and informations that have to be extracted or inserted. It should be noticed that this description is language agnostic, as everything is expressed through XML languages : messages definition (through DAML+OIL), extraction and insertion of informations (through XPath for localization and XUpdate for modifications). This global view will be broken down in relation with roles, so we will be able to *instantiate* directly this interaction within the multi-agent system.

Keeping the global view of interactions is important for designers, but to effectively add the interaction within the system, we need to define how interactions will be processed in such a way that developers should have as little work as possible. Therefore, we have defined a projection mecanism that can transform the global view of an interaction into local views for each role. Thus, an agent does not have to know the whole interaction, he just has to handle information that are relative to its role.

This projection is done in relation with roles : it is possible to build partial view of the global interaction such that agents know when a new interaction process happens, and which rules should be applied for this kind of interaction. Figure 2 illustrates the projection of the MAKEPAYMENT interaction in relation with the *Creditor* role. For this interaction, the *Creditor* knows that he has to send a request to receive credit card informations from the *Debtor*. Then, he can ask to the *Banker* to be paid, before receiving an acknowledgement.

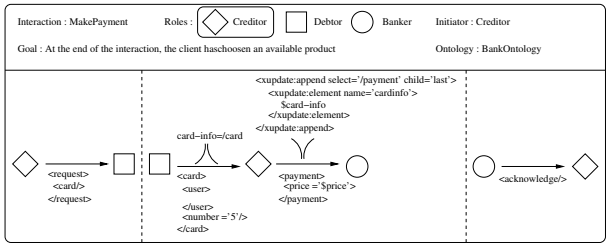

Figure 2: The payment interaction seen by the Creditor role

Despite the simplicity of this e-commerce process example, it identifies some key features : the process can be described in terms of interactions and can be kept abstract. The main advantage of such a description is that the developer keeps a global view of the interaction and delegates to the system the spreading of coordination handling code.

4. CONCLUSION

Pervasive computing is becoming a reality because of the hardware environment that surrounds people more and more. But, this availability that provides the user with multiple devices is a major outcome for designers. They have to develop ad hoc solutions that are time and money consuming, because tools and concepts that are available today do not fit. We argue that multi-agent systems concepts are the right metaphors to build pervasive systems and to address all these factors. Pervasive computing is inherently based on interactions.

Thus, multi-agent concepts like : the idea of empty agent that can evolve, the notion of skill that can be dynamically added or removed, the notion of role that allows designers to specify the behavior in terms of skills, the notion of interaction, that gives to designers a global view of messages flows between entities of the system, are the right metaphors for managing the dynamicity and complexity of communications in pervasive systems. Specifically, the notion of interaction gives to the designer a global view of role's interactions, and eases the deployment of new interactions in the system. It also vanishes the notion of application, which is replaced by an incremental development of the system.

5. REFERENCES

[1] R. Scott Cost, Ye Chen, Timothy W. Finin, Yannis Labrou, and Yun Peng. Using colored petri nets for conversation modeling. In *Issues in Agent Communication*, pages 178–192, 2000.

[2] J. S. Brown M. Weiser, R. Gold. The origins of ubiquitous computing research at parc in the late 1980s. *IBM Systems Journal*, 1999.

[3] JC. Routier, P. Mathieu, and Y. Secq. Dynamic skill learning: A support to agent evolution. In *Proceedings of the AISB'01 Symposium on Adaptive Agents and Multi-Agent Systems*, pages 25–32, 2001.