CONTEXTUAL ADAPTATION FOR UBIQUITOUS COMPUTING SYSTEMS USING COMPONENTS AND ASPECT OF ASSEMBLY

Daniel Cheung-Foo-Wo[1,2], Jean-Yves Tigli[1], Stéphane Lavirotte[3,1], Michel Riveill[1]
[1] *I3S Laboratory – UNSA / CNRS*
930 Route des Colles, 06903 Sophia-Antipolis
[2] *CSTB*
290 Route des Colles, 06904 Sophia-Antipolis
[3] *IUFM Célestin Freinet*
89 Avenue George V, 06046 Nice
{cheung,tigli,lavirott,riveill}@i3s.unice.fr

ABSTRACT

Because of the growing diversity of mobile computer terminals and communicating objects, we observe the emergence of applications using the notion of context. Firstly, we propose an analysis of definitions of context. Then, we present a model in which we simplify the expression of the problems caused by the interaction between the computer, the user and their environment. We also give the definition of the notion of proximity between entities in a multi-dimensional space and apply this notion to the previous model. Proximity of an entity refers to verify some conditions in a certain context. If the conditions are verified, some modifications and adaptations must be made at the application level. We propose an innovative software adaptation based on aspects of assemblies and dynamic reconfiguration of applications.

KEYWORDS

Software adaptation, context, aspect and software components

1. INTRODUCTION

Because of the growing diversity of mobile computer terminals and communicating objects, we observe the emergence of applications using the notion of context. This paper tackles the consequences of the mobility of systems on software development, namely the conceptual and design tools and the contextual adaptation of the final application. Mobile Computing is conceptually close to Ubiquitous Computing, which corresponds to the technical evolution explained by Weiser (1991) 16 years ago. It focuses on systems according to their location, their dynamic interconnections and the context of application (Lyytinen and Yoo 2002).

2. OUR NOTION OF CONTEXT

We present a model in which the expression of problems caused by the interactions between the computer, the user and the environment is simplified. We distinguish the processing of information from the parts responsible for interaction such as IO devices and propose to model a mobile system with two types of entities: processing units (*CPU*) and IO devices (*D*). The devices *D* are the unique way for the computer to interact with the environment. In addition, *D* can be classified as follows: D_c, responsible for the communication between the CPU and the communication network, D_i, the information storage, D_u, the user interaction and D_e, IO interface to the physical environment. The latter D_e is characterized by the data it manipulates and is consequently not used for the communication between the CPU, D_i and D_u.

A typology of research communities which study domains of Mobile Computing can be drafted. Hence, the Mobile-Agent community focuses on code mobility in an environment composed of a *CPU*, D_c and D_i. Conversely, the Context-Aware community works essentially on systems composed of D_e. This community is close to the community studying Mobility and HMI, which is interested in the *CPU*, D_u and D_e.

Entities are connected to one another by links. A link represents an access to different functionalities. We distinguish the permanent links noted L_p from temporary ones L_t. Therefore, the impact of mobility on the computer system can be seen as the introduction of L_t into the model. Mobile Computing involves systems in which different and heterogeneous temporary devices stand for the basic construction blocks. We call them "multi-devices" systems. As noticed by Jansen et al (2005), in order to improve the development of those specific applications, the Mobile Computing domain should provide a new generation of IDE. The dynamic configuration of Mobile Computing systems relies on several L_t between *CPU* and *D*, (present in the context of *CPU*). This states that every application can potentially have access to many *D* and *CPU* that have to be managed by the application itself or by other dedicated processes. So, this limits access on following criteria of authentication, ergonomy or localization. Those subtypes of L_t, which require contextual conditions to be verified are called *contextual link* L_c. A Mobile Computing system is composed of *CPU* and *D* linked by L_c.

Given those entities, we now propose to give a formal definition to the notion of context. Firstly, we distinguish a situation from a context. The situation of an entity is a set of attributes of a context which characterize a certain state in a set of spaces $\{\varepsilon_i\}$. For instance, a space ε_i can be a 2 or 3 dimensional space used for the GPS localization or the emotional state of the user. The situation of an entity is represented by a vector in a global space $\varepsilon = \mathrm{U}\varepsilon_i$. An entity e_i belongs to the context of an entity of reference e_r in a given space ε if and only if (iff) its situation is near the situation of e_r in that space. Secondly, a context is a set of entities noted $\mathrm{C}(e_r, \varepsilon)$ verifying the attribute of the space ε. Hence, there exists an L_c between the entities e_i and e_r iff $e_i \in \mathrm{C}(e_r, \varepsilon)$ in the space ε. We define what the informal expression "to be near" formally means. This is the notion of *proximity*. It is not restrictive but relevant when managing the context compared to (Pauty et al 2005). The context $C(e_r, \varepsilon_i)$ is consequently based on a cost-function noted $\mathrm{c}\varepsilon_i$ in the space ε_i such that $\mathrm{c}(x,y) \geq 0$, $\mathrm{c}(x,y) \neq 0 \rightarrow x \neq y$, $\mathrm{c}(x,y) \leq \mathrm{c}(x,z)+\mathrm{c}(z,y)$ and $\mathrm{C}(e_r, \varepsilon_i)=\{ e_i / \mathrm{c}\varepsilon_i(e_i, e_r) \leq \mathrm{C}_i \}$ where C_i is constant.

We consider that entities are in the form of a couple ({CPU,Application} - {devices}). According to available devices, well-suited adaptations are selected. Our model of the context allows us to design filters which aim at verifying the applicability of *contextual schemas*. Thus, according to the state of the devices e_i, each filter verifies whether the devices are near the CPU e_r given a space ε_i. A contextual schema is applicable in relation to an entity of reference e_r iff the needed entities successfully pass through each filter.

3. SOFTWARE ADAPTATION

Some works propose techniques for the extraction and representation of data in order to be used in application adaptation. The trend has been to build a generic representation of the execution context. Chronologically, Bellavista et al (2003) proposed to manage security for mobile agents by using metadata in profiles and authorization rules. Mahéo et al (2004) proposed environments (RAJE, SAJE) to grant access to system resources to Java applications. David and Ledoux (2006) described the execution context as a set of contextual domains such as physical resource, network topology or geophysical context (temperature and location). However, the propositions to take the user into the context is still realized in an ad-hoc way by redefining the context for every new application (Want et al 1992; Abowd et al 1997).

3.1 Minimalist Software Model

Our approach consists in adapting applications written under the form of components' assembly according to the context, a work close to (Rashid and Kortuem 2004) who used *classical* aspects for adaption to pervasive environments. Our approach remains compatible with a layered approach. But we underline the cross-layered modifications of the assembly of components. We make use of the separation of concerns in terms of contextual schemas which stands for *contextual* aspects of assembly. We work firstly on the adaptation of the application to wide contexts such as security, localization and emotional state and secondly, on contextual

schemas described below. A contextual schema is divided into two parts: contextual areas and an aspect of assembly. This technique allows the simultaneous application of many aspects of assembly.

3.1.1 Component Model

We present in this part the component model of the application based on the minimalist framework described by Cheung-Foo-Wo et al (2006). The purpose of this framework is to offer a rapid development environment using software components. Thus, a component has input and output ports. Outputs can be dynamically bound to inputs. When we deal with dynamic reconfiguration, we mean that components are added or removed and ports are bound or unbound during runtime. This model is composed of two kinds of components: pure software components and "mixed" components. Mixed components are software components depending on one or several IO devices viewed as external resources. The implementation framework allows to write applications following this model in the various languages such as Java, C#, Objective-C and C++. At the implementation level, outputs emit events (event mechanism as referred in C# language for instance). Some inputs are subscribed to some outputs. We represent the event mechanism by appending sequence-components to the output of a component. This component model is based on the JavaBeans model (Hamilton 1997). The aim is not to be different from the existing approaches that consist in simplifying the programming of microcontrollers as proposed by Gay et al (2003) or bringing together aspects and components as David et al (2006) do. The aim is to offer a model suited to the use of IO devices.

An application consists in (one or) several software components which are interconnected which forms the assembly of components. Cheung-Foo-Wo et al (2006) provide a tool to manage and isolate the reconfiguration of assemblies called "Container". Other tools, called "Designers" are front-ends used to manage the assemblies through the Container. Each designer has its own representation of the application. Namely, we detail the model of the aspect-oriented designer, which enables to run the application under the form of aspects of assembly. These tools can either be standalone software or components. Container and Designer communicate through an API composed of few main entries: add or remove links or components and obtaining the assembly configuration. Architecture description languages are often used to describe assemblies. The nesC approach of Gay et al (2003) allows the programmer a description of assemblies throughout complex connectors expressed in "configurations" completing the components called "modules". However, those approaches do not take into account the superposition of assemblies according to an explicit semantics based on a set of logical rules of the aspect weaver as detailed in the next section.

3.1.2 Aspect of Assembly Model

Our software adaptation is based on aspects of assemblies and dynamic composition of components. The technique of applying aspects to assemblies is made out of a weaver algorithm, which results in a set of modifications of the assembly. Aspects of assembly are written using the ports of components. They are modeled in terms of a function returning modifications for reconfiguration. This function takes as parameters a list (p_1, …, p_n) of descriptions of components. Let O be the set of the lists of reconfiguration:

$$s_{e_r} \begin{cases} P^n \rightarrow O \cup \{nil\} \\ (p_1,..., p_n) \rightarrow reconfiguration \end{cases}$$ where $o \in O$ if the schema is applicable.

An aspect $s_{er}=(p_1, \ldots, p_n)$ is applicable according to a reference entity e_r if for each entity e_i corresponding to the component p_i which are mixed, every e_i is in the contextual area of e_r. For each notification of e_r and for each aspect s_{er}, we consider two cases: s_{er} is either already applied and not applicable anymore, or not yet applied and applicable. In the first case, the modifications operated by the previous s_{er} (if it exists) during the last transformation are undone. In the second case, the modifications returned by s_{er} are operated. For all other cases, no action is taken. We may unfortunately identify a problem in this algorithm. Undoing the transformation can be complex and may depend on the history of every applied aspect. The model is based on the Interaction Specification Language (ISL), which describes the interactions between components implemented as objects (Blay-Fornarino et al 2004). It specifies how to merge different interaction schemas into a single one. The algorithm of the aspect weaver works like the ISL merging algorithm. In this model, we have two ways to modify the behavior of an application: (1) by inserting subassemblies in front of input ports or inserting subassemblies after an output port. Here is the simplified grammar of an aspect:

<pointcut> : <component>.<output> | <component>.<input>
aspect <name> (<component>…) : <pointcut> { <programmed behavior> }

The behavior can be expressed using the following operators: *sequence* of two subbehaviors, *parallelism* between two subbehaviors, *conditional execution* of a subbehavior, *call* and *delegate*. The two last operators are composition specific and respectively mean weak and strong composition when merging behaviors. These two keywords control the way the behaviors of the aspect are to be merged. The strong compositional keyword absorbs every composed behavior whereas the weaker makes way for the others. Those keywords are similar to the keywords *before*, *around* and *after* from AspectJ which is the reference in Aspect-Oriented Programming. However, the latter applies to advices while the first applies to behavioral operators.

3.2 Example

Aspects' being applied integrates devices into an application. An aspect is selected when a set of entities (devices though mixed components or pure software components) are near the reference entity which is the user application (CPU). The components used for this application are a Hi-fi system, a TV, a clock, a wireless joystick and a Head Mounted Display (*hmd*). The entity of reference is the joystick noted *j*.

3.2.1 Definitions

Below you can see a simplified description of the different components:

mixed component Joystick: **output** Clicked(), Down(), Up(), Left(), Right().
mixed component Tv: **input** On() { … }, Off() { … }.
mixed component Hifi: **input** OnAndPlay() { … }, Off() { … }.
software component HeadMountedDisplay: **output** Refresh(), **input** Print(msg) { … }.

Contextual schema of the TV: csTV

aspect *csTV*(*joystick*, *hmd, ecrantv, clock*):
hmd.^Refresh() { *call* ; *hmd.Print*("Click for TV on.") ; *hmd.Print*("Down for TV off.") }
joystick.^Clicked() { *call* || *tv.On()* } ***joystick.^Bas***() { *call* || *tv.Off()* }

The first rule specifies that when the head-mounted display is refreshed by a component (which we do not describe here), the compositional weak point is set, and then we successively display two messages to inform the user of the functionalities offered by the TV. The second rule says that when messages are sent from the output "Clicked" of the joystick, there is a compositional weak point and concurrently, the TV is turned on. The third rule says that when messages are sent from the output "Down" of the joystick, there is a compositional weak point and concurrently, the TV is turned off.

Contextual schema of the Hi-fi system: csHifi

aspect *csHifi*(*joystick*, *hmd, hifi*):
hmd.^Display() { *call* ; *hmd.Print*("Click to play a song.") ; *hmd.Print*("Down to turn off.") }
joystick.^Clicked() { *call* || *hifi.OnAndPlay()*} ***joystick.^Bas***() { *call* || *hifi.Off()* }

3.2.2 Calculus

We focus now on the consequence of being in the context of the components cited before. We calculate the proximity of the different objects and see if aspects apply. If both do, their behavior is to be composed and finally, we describe the transformation of the resulting aspect to a set of reconfigurations of the application.

The context is set as follows: the joystick is near the Hi-fi system and the TV; the time is right in the expected range and the emotional state of the user is normal. The two contextual schemas are then selected. We have then a contextual space such as ε=U($\varepsilon_{location}$, ε_{time}, $\varepsilon_{emotion}$). The context is defined as follows: C(e_j, ε) = { e_i / cε(e_i, e_j) ≤ C }. We have a 3D global space composed of the 2D location ({x, y} in meter) relative to an arbitrary origin, the interval ({start, stop} in hour) and the emotion (from 0% to 100% - where 50% is a normal state). We can have for instance the following values for the constant C and the vector e_j of the joystick: C=(l={x:5m,y:5m},t=0h,em=5%) and e_j=(l={x:100m, x:50m}, t={start:7h, end:9h}, e_m=50%). The proximity function is given as follows for each dimension of the global space: c$\varepsilon_l(a,b)$ = {$|a_x - b_x|$,$|a_y - b_y|$}, c$\varepsilon_t(a,b)$={($|a_{start} - b_{start}|$) + ($|a_{end} - b_{end}|$)}, c$\varepsilon_{em}(a,b)$ = $|a-b|$. And the result of the merge of the two aspects is:

aspect *result* (*joystick*, *hmd, hifi, tv, clock*):
hmd.^Refresh() { *call* ; (*hmd.Print*("Click for the TV on.") ; *hmd.Print*("Down for the TV off."))

|| (*hmd.Print*("Click to play.") ; *hmd.Print*("Down for off.")) }
joystick.^Clicked() { *call* || *hifi.OnAndPlay()* || *tv.On ()* } ***joystick.^Down***() { *call* || *hifi.Off()* || *tv.Off()* }

This aspect is transformed into a set of reconfigurations of the assembly of components. The modifications can be summarized by stating that this final aspect results in (1) the connection of the *joystick* outputs "Clicked" to the inputs "OnAndPlay" of *hifi* and (2) "Down" and the connection of "On" of *tv* to the input "Off" of *hifi* and *tv*. Finally, the "Refresh" output of *hmd* is connected a list of sequential components and other specific components. It appends the different messages to the parameter of the input "Print".

There is still a limit related to the use of the ISL weaver for the aspect: the problem of the parallelism and its semantics. Indeed, the result of the merge of two aspects consists in assigning two functionalities: play a song and turn on the TV. Those two functionalities are assigning in parallel according to the merging process of ISL. However, the semantics of the parallelism is delegated to the executing framework. If the expected behavior is not a multithreaded execution but simply a priority, we have a problem. In that case, we need a model for scheduling this resource and decide for a final semantics for the operator.

4. CONCLUSIONS

Mobile Computing presents new features. The heterogeneity of components should be considered so that alternative devices potentially unworn could be used as in Wearable Computing. We have proposed a model of the Interaction Context (System, User and Environment) and an approach to take into account the context in order to adapt a structured software application by weaving aspects of assembly. This approach is different from usual approaches as it defines and uses a unified context and describes a cross-layered software adaptation through logical aspect weaving. Limits still exist and are caused by the current semantics of the aspect of assembly language. Future works will explore different semantics using domain specific languages.

REFERENCES

Abowd G. D. et al, 1997. Cyberguide: A Mobile Context-Aware Tour Guide. *Baltzer/ACM Wireless Networks*, Vol. 3, pp. 421-433.

Bellavista P. et al, 2003. COSMOS: A Context-Centric Access Control Middleware for Mobile Environments. *MATA: Mobile Agents for Telecommunication Applications*, Vol. 2881, pp. 77-88.

Blay-Fornarino M. et al, 2004. Software Interactions. *Journal of Object Technology*, Vol. 3, pp. 161-180.

Cheung-Foo-Wo D. et al, 2006. Wcomp: a Multi-Design Approach for Prototyping Applications using Heterogeneous Resources. *Proceedings of the 17th IEEE International Workshop on Rapid System Prototyping*, Chania, Crete.

David P.-C. and Ledoux T., 2006. An Aspect Oriented Approach for Developing Self-Adapting Fractal Components. Proceedings of The Software Composition, Satellite Event of ETAPS, Vol. 4089.

Gay D. et al, 2003. The nesC Language: A Holistic Approach to Networked Embedded Systems. *Proceedings of Programming Language Design and Implementation*.

Hamilton G., 1997. JavaBeans. *API Specification from Sun Microsystems*, Version 1.01.

Jansen E. et al, 2005. A Programming Model for Pervasive Spaces. *Proceedings of the 3rd International Conference on Service Oriented Computing*, Amsterdam, The Netherlands.

Lyytinen K. and Yoo Y., 2002. Issues and Challenges in Ubiquitous Computing. *Communications of the ACM*, Vol. 45, No. 12, pp. 62-65.

Mahéo Y. et al, 2004. Middleware Support for the Deployment of Resource-Aware Parallel Java Components on Heterogeneous Distributed Platforms. *Proceedings of the 30th Euromicro Conference - Component-Based Software Engineering Track*, pp. 144-151.

Pauty J. et al, 2005. *Using Context to Combine Virtual and Physical Navigation*. Technical Report 5496, INRIA, Rennes.

Rashid A. and Kortuem G., 2004. Adaptation as an Aspect in Pervasive Computing. *Proceedings of the Symposium on Principles of Distributed Computing*, Vancouver, Canada.

Satyanarayanan M., 1996. Fundamental challenges in mobile computing. *Proceedings of the Symposium on Principles of Distributed Computing*, pp. 1-7.

Want R. et al, 1992. The active badge location system. *ACM Transactions on Information Systems*, Vol. 10, pp. 91-102.

Weiser M., 1991. The Computer for the Twenty-First Century. *Scientific American*, Vol. 1, Paper, pp. 94-104.