A Dynamic mechanism for solving Interfering Adaptations in Ubiquitous Computing Environment

Sana Fathallah Ben Abdenneji, Stéphane Lavirotte, Jean-Yves Tigli, Gaëtan Rey, Michel Riveill

A Dynamic mechanism for solving Interfering Adaptations in Ubiquitous Computing Environment

Sana Fathallah Ben Abdenneji[1], Stéphane Lavirotte[1], Jean-Yves Tigli[1], Gaëtan Rey[1], Michel Riveill[1]

[1]I3S Laboratory, 930 Route des Colles, 06903 Sophia-Antipolis France

{fathalla, stephane.lavirotte, jean-yves.tigli, gaetan.rey, michel.riveill}@unice.fr

Abstract. Dynamic adaptation is a central need in Ubiquitous computing. In this kind of system, it is frequent to see the appearance of interference, when there are several adaptations to be applied on the application. In this paper, we present an automatic mechanism for dynamic interference resolving. Applications and adaptations will be represented by graphs; then we apply graph transformation rules to compute at runtime the solution.

Keywords: self-adaptation, software composition, interference resolution, graph transformation.

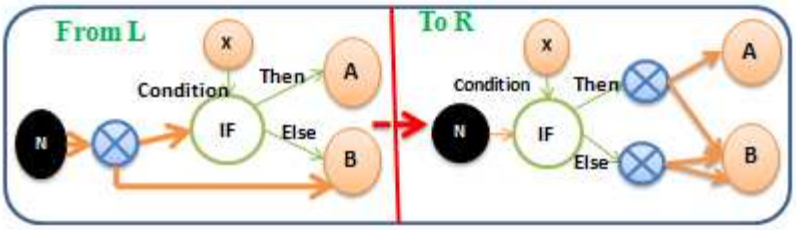
1 Introduction

Ubiquitous computing relies on objects of our daily life. These devices are able to communicate together and they constitute the software infrastructure, on which the ubiquitous system is based. In such kind of systems, applications are designed based on components assembly or orchestration of services. They can be represented using ***graphs*** where *nodes* are the entities (components or services) and *edges* model the interaction between these entities. Because of devices mobility, the graph modeling the application must be continually adapted to consider the changes of the environment (appearance/disappearance of devices). So, the system should react *automatically* and *transparently* by applying adaptations. An adaptation, which can also be modeled as a graph, specifies behaviors to be integrated into the initial graph (application). As a consequence, an adaptation changes the graph structure of the application by adding or modifying links and/or nodes. In addition, adaptations have to be written without *a priori knowledge* of the other adaptations of the system. So, during the step of adaptation, it is possible to have some modifications which can be attached in common points of the graph: we call these ***interferences.***

Researchers in self adaptation applications have used different techniques to deal with interferences. Zhang *et al.* [2] specify adaptations precedence at modeling level. However, interferences occur at runtime; so the resolution must be applied at runtime. Dinkelaker *et al*. [1] propose to dynamically change the adaptation composition strategies according to the runtime state of application. In this approach, if we add a new adaptation to the system, the developer should study its relationship with the other adaptations, which is a complex task. From these works, we can conclude that the proposed techniques cannot remove all interferences. To alleviate this problem, we propose to merge automatically interfering adaptations without preventing interferences explicitly. Our mechanism guarantee the independence between adaptations that can be composed whatever their order, and that can be added or removed easily to the system.

2 Merging interfering adaptations using Graph Transformation

Since adaptations are designed independently, they can interfere. Our solution is to merge them. This is possible due to the knowledge of the semantic of some nodes of the graph. Our graphs have two classes of nodes. **Whitebox** nodes (are operators that explain their semantics) and **Blackbox** node (are devices; they encapsulate the functionalities that can be only accessed by their ports, without knowing their semantics). The interference resolution process can be divided into two steps. The first step is the interference detection. The input of this step is the graph G which is obtained from the superposition of all graphs of adaptations on the graph of the initial application. On the graph G, we add a specific component ⊗ (*Fus*) to mark all interference problems. The second step is interference resolution. Since we work at graph level, the resolution will be a transformation of the graph G to the graph G'. Therefore, we need to define graph transformation rules that specify how to merge all known semantic nodes. We have defined a set of merging rules which derived from previous works [3]. A graph transformation rule has the form of $r:L \rightarrow R$. It can be performed if there is an occurrence of L in the host graph G. The rule execution implies to: (1) remove L and (2) add the R graph. The figure 1 represents an example of the graph transformation rule. It shows the merging of the semantic known operator IF (the conditional behavior) and a message send. The conditional behavior "*IF*" is specified by three parts. "*X"* node represents the condition to be evaluated. When "X" is ***True***, we execute the node *"A*", otherwise "*B*" will be executed.



Our composition mechanism is independent from application's implementation because it occurs on the graph G, which abstracts all details. The main property of our composition is the ***symmetric***. This property consists of three sub-properties: *associativity*, *commutativity* and *idempotency*. It means that there is ***no order*** in which composition process should be applied. As a consequence, adaptations are independent of each other and it can be composed in an unanticipated manner. These properties allow the adaptation process to be deterministic. To enable the merging of these interfering rules with the previous properties, we constrain the adaptation language. Whatever the language used to write the adaptation, it must be based on a limited set of operators with a well-known semantic that can be merged. In our implementation, 6 operators were defined; which implies the definition of 16 graph transformation rules.

References

1. Dinkelaker T., Mezini M., and Bockisch C.: The art of the meta-aspect protocol. In Proceedings of the 8th ACM international conference on Aspect-oriented software development, pp. 51–62. ACM, (2009).
2. Zhang J., Cottenier T., Van Den Berg A., and Gray J. : Aspect composition in the motorola aspect-oriented modeling weaver . In Journal of Object Technology , (2007).
3. Tigli J. Y., Lavirotte S., Rey G., Hourdin V., Cheung-Foo-Wo D., Callegari E., and Riveill M. WComp Middleware for Ubiquitous Computing: Aspects and Composite Event-based Web Services. In Annals of Telecom, pp. 197-214, (2009).