MergeIA: A Service for Dynamic Merging of Interfering Adaptations in Ubiquitous System

Sana Fathallah Ben Abdenneji, Stéphane Lavirotte, Jean-Yves Tigli, Gaëtan Rey, Michel Riveill
University of Nice Sophia-Antipolis
Nice, France
{fathalla, stephane.lavirotte, jean-yves.tigli, gaetan.rey, michel.riveill}@unice.fr

***Abstract*— The composition of adaptations with system's application does not always yield to the desired behavior. Each adaptation occurs correctly when it is separated but it may interact with other adaptations when they are combined. These interactions can affect the final behavior after adaptation; we call this an *interference*. This paper presents an on-going work, which aims to build a generic approach for the dynamic resolution of adaptation interferences in ubiquitous applications. We represent application and adaptation details by graphs; then we apply graph transformation rules on these graphs to resolve interferences. This allows us to express our approach independently of any implementation details of applications and adaptations.**

Keyword-*software composition; self-adaptation; interference resolution; graph transformation.*

I. INTRODUCTION

Nowadays, ubiquitous systems are present in several environments. In these cases, the user does not have to carry out most actions; the system reacts automatically and transparently to its changes. The computing facilities in ubiquitous system are used to anticipate user needs and to make information being available anywhere and at anytime. The goal of this work is to create applications in ubiquitous computing environment. Generally, software application relies on processing units that interact together. Lot of programming paradigms produce ubiquitous applications (component based), which can be represented using graphs where *nodes* are the processing units and *edges* are the interactions between these units. In ubiquitous computing, some processing units are embedded on sensors and mobile devices of our everyday life. These devices constitute the software infrastructure, on which the ubiquitous system is based. Indeed, the functionalities of such devices, which are generally managed as software services, may unexpectedly appear or disappear. Therefore, ubiquitous systems must be adapted to these infrastructure changes. Due to the mobility of devices, we cannot *anticipate* in advance which adaptation will be applied. Therefore, adaptation should be independent of each other, which allow them to be applied without *a priori knowledge* of other adaptations. In ubiquitous computing, infrastructure changes occur during execution; so, application should be adapted at runtime: it is the *dynamic adaptation* [1]. Our adaptation acts on application graph by adding and/or deleting edges and nodes (Figure 1).

Problem: In this paper, we focus on dynamic adaptation of applications to their infrastructure changes. We have seen that adaptations should be independent of each other. So, when they are composed with the graph of the initial application, ***interferences*** may occur. There are several definitions of interference. In our work, we detect interference if two (or more) adaptations try to modify a common point in the graph of the initial application (by adding and/or deleting edges and/or nodes). So, interfering adaptations share edges and/or nodes together and with initial graph.

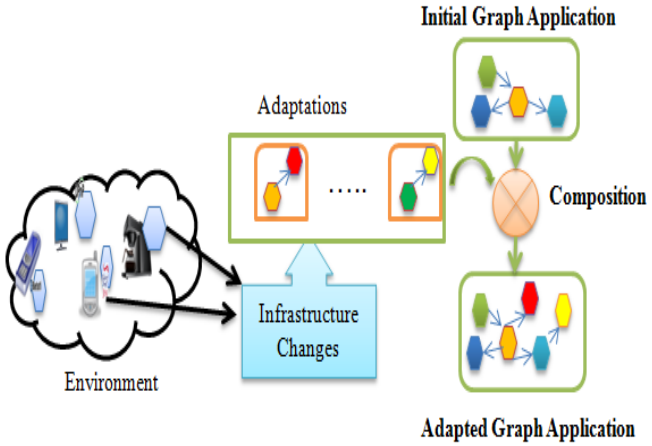

Figure 1. Dynamic adaptataion of ubiquitus application.

Scenario

In this paper, we will use the following scenario to illustrate the problem of interference between adaptations. We expose also the process of our approach through this example. **"***The increase of energy cost encourages the use of an optimizing policy. For this purpose, Nathalie uses in her house a system of* ***intelligent power management****. The first adaptation occurs when she enters her house. The system would enable the switches to open the shutters if the outside brightness is sufficient. Otherwise, it turns on the light. Nathalie lives with her grandmother who has vision problems. When the grandmother enters a room, the system will turn on the light".*

When Nathalie enters with her grandmother the system will be in interference because no priority has been specified between users (one cannot know all users in a ubiquitous environment). If there is enough brightness outside, the system opens the shutters for Nathalie? Turns on the light for

 ISBN: 978-1-61208-171-7

the grandmother? Or will it do the two actions? "*In addition, Nathalie uses in her house special light. When the light receives an event it will inverse his state"*. If there is not enough brightness outside the light receives two events. The system will send an event to turn on the light for *Nathalie* and another event to turn on the light for *the grandmother*. The light will be turned off despite there is not enough outside brightness. How to solve these interferences?

The paper is organized as follows: next section briefly describes some related work to detect their limits. In Section 3, we introduce our approach to identify and resolve interferences; we also apply our solution on the previous example. In Section 4, we present implementation details and we evaluate the response time of our approach. Finally, Section 5 concludes and opens the way for future works.

II. RELATED WORK

Despite the independence between adaptations, some interference may occur when they are composed. Baresi *et al.* [8] focus on the ability of dynamic reconfiguration of SOA (Service Oriented Architecture) application using graph as platform abstraction model. They propose several adaptations at graph level (adding/deleting nodes and links). In their approach, there is no interference because they define explicitly the order of applying adaptation. However, in the field of ubiquitous computing, we cannot predict which adaptation will be applied because it depends on infrastructure changes due to mobility for example.

Other works [5] [7] [11] focus explicitly on the problem of adaptation interference detection. Ciraci *et al.* [11] use a graph formalism to identify interference. Graphs represent the several states of a program according to different order of adaptations applying. They detect interference if the final state changes according to the order of adaptation applying. The motivation of Whittle e*t al.* [5] and Mehner *et al.* [7] was the early detection of interference within the software engineering process. To find potential inconsistencies, they analyze adaptation interactions at the level of requirements modeling. To do that, they use graph transformation technique since it includes a mechanism called Critical Pair Analysis [10]. This mechanism allows interference detecting. However, it is not enough to detect interference without suggesting a resolution.

In order to address this limitation, Zhang *et al.* [6] proposes an explicit approach to resolve interference at design time. They describe how adaptation precedence (before, after) can be specified at modeling level in order to produce correct behavior. So, they reduce interference before proceeding to the implementation. However, the application in ubiquitous computing field needs runtime interference resolution.

Runtime resolution of adaptation interference was proposed by Greenwood *et al.* [9]. They investigate a solution to interference in the context of AO-Middleware platform. To do that, they define "*interaction contract"* which are used at runtime to assure that interference does not occur. These contracts express several strategies to resolve interferences such as priority and precedence and logical operator (to combine contracts). Despite the use of these contracts at runtime, the specification is made by the developer who must include all dependent relationships between the adaptations. If an automatic resolution is not possible, a notification is sent to the developer to include this case into the contract.

The strategy of interference resolution may depend on the runtime state of application. Dinkelaker *et al*. [12] propose to dynamically change the composition strategies according to the application context. They define an extensible ordering mechanism which can be adapted at runtime. This type of approach is not suitable for ubiquitous computing because we should specify at design time the relationship between all adaptations according to different context state. If we add a new adaptation to the system, the developer should study its dependence with the other adaptations and also the context, which is a complex task with a high combinatorial.

Through the study of works, it is clear that there is no implicit approach for solving interference without developer's intervention. Our proposed approach is to merge interfering adaptation without preventing interferences explicitly. We guarantee independence between adaptations that can be composed whatever their order, and that can be added or removed easily to the system at runtime. The first work on this subject was developed in our team [2]. The essential contribution was the definition of the composition mechanism, which includes interference resolution process. The composition mechanism is limited to the language defined in [2]. It is very difficult to extend it to support new known semantic operators due to an implementation with an inference engine in Prolog. In addition, the representation of the adaptations is not homogeneous with the representation of the application. The adaptations are specified in the language but we work on assemblies of components which are represented as graphs. Therefore, it is necessary to make two transformations from graph to the language (syntactic tree), then from the language to the graph form. We think that it would be relevant to remain closer to the execution model (i.e., the level of the graph forms). Then, it could be interesting to explain these adaptations as graphs. The use of graphs will allow us to have a mechanism of interference resolution independent from the language of adaptations.

III. GLOBAL APPROACH

The aim of this research is to provide automatic adaptation interference composition that replaces the mechanism of precedence. The composition process occurs at runtime and is independent of adaptations that are in interference.

A. Process of interference resolution

The process of our approach is given in Figure 2. Each adaptation is represented as a graph. All graphs will be superposed to the graph of the initial application. We obtain a graph G, which represents the application of all possible adaptations on initial graph. Our composition mechanism is independent from application's implementation because it occurs on graph G, which abstracts all details.

The first step is the interference detection process. Since adaptations are independent; they can interfere each other.

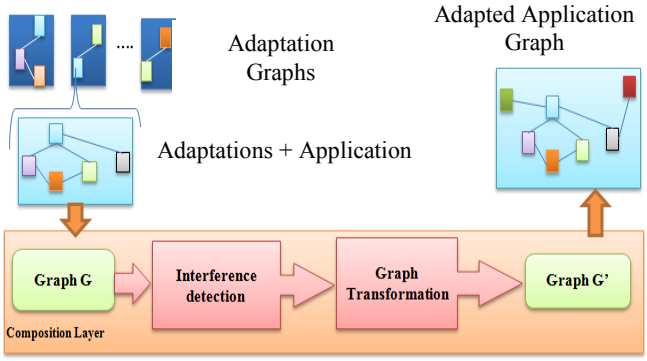

Figure 2. Composition process for interference resolution

So, we add a specific component ⊗ (Figure 3) to mark these points in order to check off interference. In the scenario presented above, we have two adaptations: one for Nathalie and another one for her grandmother. When Nathalie goes into home with her grandmother, these adaptations will be applied. The interference is presented in Figure 3.

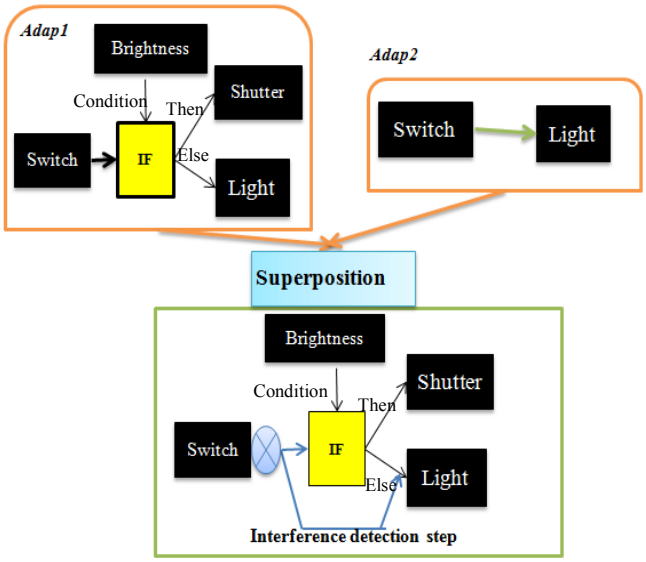

Figure 3. Graph transformation rule for conditionals merging

Next step is interference resolution. Since we work at graph level, the resolution of interference will be a transformation of this graph G to a new graph G' where all problems were resolved. Therefore, we need to define graph transformation rules that specify how the problem will be resolved.

B. *Graph transformation and type Graph*

The rewriting of a graph G into a graph G' is a substitution of a subgraph *L* of G by a subgraph *R*, where *L* is the left-hand side of the rule and *R* is the right-hand side. Therefore, a rewrite rule has the form of $p:L \rightarrow R$ and is applicable to a graph G if there is an occurrence of L in G. The application of the rule implies to: (1) remove the graph L and preserve the graph L∩R (is the graph part that is not changed.) and (2) add the graph corresponding to R∩ (L \ R) (define the part to be created).

To apply graph transformation rules, we have to define the ***type graph***. In our graph, we have two classes of nodes: **Blackbox** nodes represent devices. They encapsulate the functionalities that can be only accessed by their ports, without knowing their semantics. **Whitebox** nodes partially explain their semantics.

To define node, we need to specify two attributes: *CN(n)* is the node identifier and *CTy(n)* is node type (blackbox or whitebox). Graph's edges represent interaction between nodes. On the edge, we specify a label to indicate the semantic of interaction (for example, the conditional behavior IF has three parts, so we put on the outgoing arc one of the three following labels: *Condition*, *Then*, *Else*).

C. *MergeIA: Merging Interferering Adaptataion*

Until now, we have identified the interference between adaptations. Our approach is to merge adaptations that interfere and not to explicitly prevent interferences. Therefore we propose the merging of adaptations from the knowledge of the semantics of *Whitebox* nodes using graph transformation rules. This role is attributed to *MergeIA* service.

MergIA (Merge Interfering Adaptation) service includes several graph transformation rules which define how to merge all known semantic nodes. We defined a set of merging rules which derived from previous works [2]. Our composition is ***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. It allows adaptations to be independent of each other and that they can be composed in an unanticipated manner. Therefore, these properties allow the weaving process to be deterministic.

We continue with the defined scenario. To resolve the identified interference, *MergIA* uses the graph transformation rules for the merging of the conditional behavior IF and a message (Figure 4).

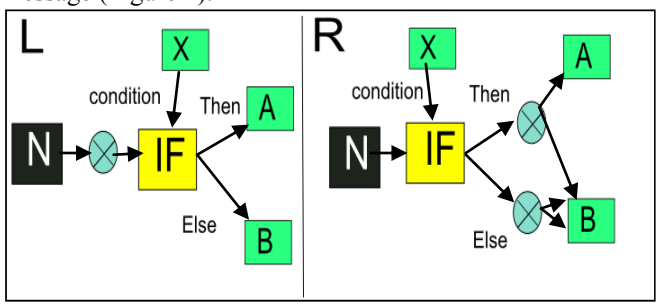

Figure 4. Graph transformation rule for conditionals merging

The conditional behavior "*IF*" is specified by three parts. "*X*" node represents the condition to be evaluated (in our scenario *X* is unified to *blackbox node* ***Brightness***). When this condition is ***True***, we execute the node "*A*" (the message ***open*** the ***shutter***). Otherwise "*B*" will be executed (***turn on the light*** *for Nathalie*). When two adaptations add two bindings to blackbox node "*N" (Switch)*, (binding to IF behavior and binding to a message in L graph), the result of the merging operation consists in the duplication of the

message "*B"* into the two sub part of IF behavior (*Then* and *Else* in graph R). Therefore, we propagate the merging operator ⊗ (Figure 4) and we obtain two merging operation. The first operation is the merging of the node "*A"* and "*B"*. The second operation merges "*B*" and "*B*". This propagation allows other rules to be applied according to the semantic of nodes "*A"* and "*B"*. In our scenario "*A"* and "*B"* are method calls (message). The result of the merging of two different messages consists to add a ***parallel (PAR)*** operator between the two bindings. The merging of the same message produces a single link to this message. The interference resolution step produces the graph of application given in Figure 5.

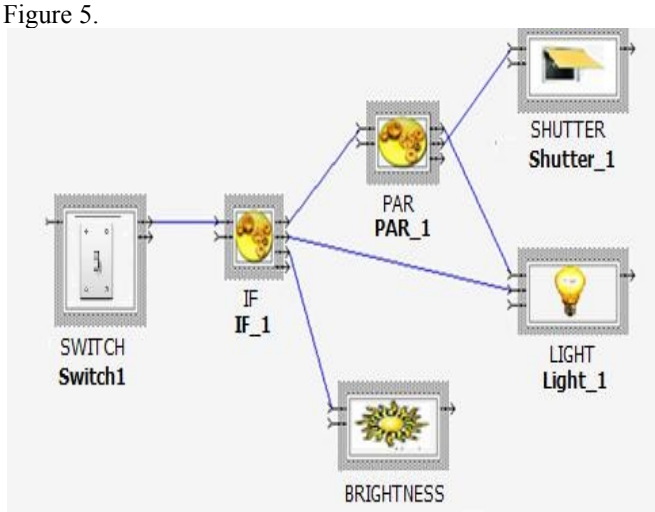

Figure 5. Final application after interfering adaptations resolution

In Figure 5, if there is enough outside brightness, we turn on the light for the grandmother in parallel with the opening of shutter. After that, the decision is left to the grandmother. She can turn off the light if there is enough brightness for her. Else (if there is not enough outside brightness) the light will be turned on (we send a single event to the light). So we solved the interference problem defined in our example scenario.

IV. IMPLEMENTATION AND EVALUATION

In our implementation, we consider service-oriented middleware [14] in order to manage heterogeneity of the devices included in the infrastructure of an application. Each application is embedded into a service which is orchestrated using component assemblies [13]. The appearances and disappearances of services are directly implemented in the appearance and disappearance of components in the platform [4].

Our approach for adaptation interference resolution was implemented as service *MergeIA*. If we detect interference, *MergeIA* receives the XML (Extensible Markup Language) description of the graph of the application. To resolve interference, it uses the graph transformation rules defined in his rule database. We defined five known semantic nodes. Therefore, we have 16 rules in the rule Database (due to the property of symmetric defined above). The graph transformation rules used in this paper can be formulated using several tools. In fact, we have used AGG (Attributed Graph Grammar System) [3] to carry out the transformations. As a consequence, we use its algorithms to resolve interference.

The complexity of the current implementation is closely related to AGG because most of the composition time is passed into the resolution interference step. The most complex operation during the application of a transformation rule is the search of a match in the graph (find an occurrence of L in G). The complexity of this operation is $O(2^{NNode})$ with *NNode* is the number of node in the left-side graph of the rule to apply. If we execute one transformation rules to resolve each interference the complexity of MergeIA will be: $O(NbInterf*2^{NNode})$. From this complexity, we can deduce the following mathematical model: $R = a1 \sum_{k=1}^{NbInterf} 2^{nk} + a2$; where n_k is the number of L graph node of the rule to apply, *nbInterf* is the number of interference, a1 and a2 are the parameters of the model and R is the duration of the interference detection and resolution.

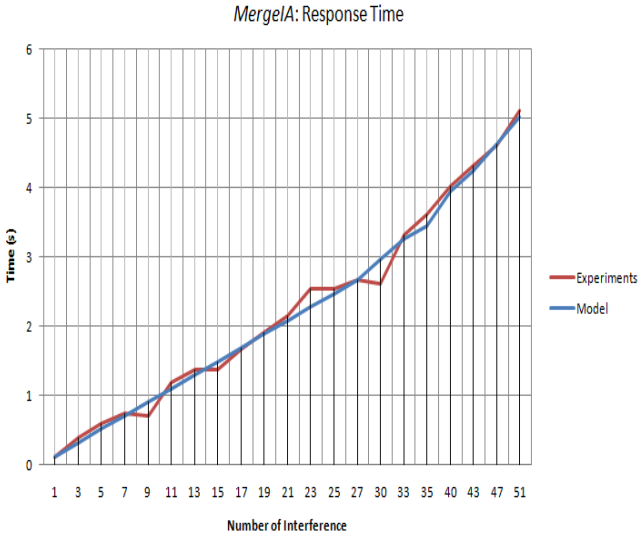

Figure 6. MergeIA: Response Time according to number of interference

We evaluated our approach in term of performance with some experiments on the duration of the interference resolution step over components assemblies randomly generated. They were conducted on a standard personal computer (Intel® Core TM2, 3GHz). For this purpose, various types of components have been instantiated randomly at runtime, in order to randomly activate two adaptations (described above in Figure 3). Our experiments involved a set of instances of adaptation, with their cardinality ranging from 0-100. The number of considered interference ranged from 0 to 50. Several experiments were made, and the Figure 6 provides a comparison between the mathematical model and experimental values. From these experiments, we can extract the following values for the model: a1=0,446 and a2=0,02 10^{-3}.

V. CONCLUSION AND FUTURE WORK

In this paper, we presented an approach for application's self-adaptation in ubiquitous computing domain. We proposed a general mechanism to resolve interference that can

occur between adaptations. The solution proposed is to merge the interfering adaptations. This is possible thanks to known semantic operators. Whatever the formalism chosen to specify adaptations, the merger considers them as graphs to automatically compute the solution. To do this, the *MergeIA* service uses graph transformations mechanism.

Our future work will be to study how we can add new semantics and extend *MergeIA* service. Actually we consider only output port to detect interferences because our defined operators have a single input port and multiple output ports. If we introduce new operators with multiple input ports we will be able to resolve interference at input port of components. Therefore we will obtain a general approach that considers two interference cases: Output and Input port.

ACKNOWLEDGMENT

This work is part of the Continuum Project (French National Research Agency) ANR-08-VERS- 005.

REFERENCES

[1] A. Rasmus, I. Schaefer, M. Trapp, and A. P. Heffter. “Component-based modeling and verification of dynamic adaptation in safety-critical embedded systems”. In Journal ACM Transactions on Embedded Computing Systems (TECS) Volume 10 Issue 2, 2010.

[2] J. Y. Tigli, S. Lavirotte, G. Rey, V. Hourdin, D. Cheung-Foo-Wo, E. Callegari, and M. Riveill. “WComp Middleware for Ubiquitous Computing: Aspects and Composite Event-based Web Services”. In Annals of Telecom, pp. 197-214, 2009.

[3] G. Taentzer. “AGG: A graph transformation environment for modeling and validation of software”. Lecture Notes in Computer Science, vol. 3062, pp. 446–453, 2004.

[4] H. Cervantes and R. S. Hall. “Autonomous adaptation to dynamic availability using a service-oriented component model”. In Proceding of the 26th International Conference on Software Engineering. pp. 614-623. USA , 2004

[5] J. Whittle, P. Jayaraman, A. Elkhodray, and A. Moreira. “Mata: A Unified Approach for Composition UML Aspect Models Based on Graph Transformation”. In Transaction on AOSD V. p191-237, Berlin 2009.

[6] J. Zhang, T. Cottenier, A. Van Den Berg, and J. Gray. “Aspect composition in the motorola aspect-oriented modeling weaver” . In Journal of Object Technology , 2007.

[7] K. Mehner, M. Monga, and G. Taentzer. “Analysis of aspect-oriented Model Weavings”. In Transaction on AOSD V. p235-263, Berlin 2009.

[8] L. Baresi, R. Heckel, S. Thöne, and Daniel Varro. “Style-based modeling and refinement of service-oriented architectures A graph transformation-based approach”. Special Issue Paper in Software and Systems Modeling Volume 5, Number 2, 187-207.

[9] P. Greenwood, B. Lagaisse, F. Sanen, G. Coulson, A. Rashid, E. Truyen, and W. Joosen. “Interactions in AO middleware”. In Proceding of Workshop on ADI, ECOOP, 2007.

[10] R. Heckel, J. Kuster, and G. Taentzer. “Confluence of typed attributed graph transformation systems”. In Journal Graph Transformation, pp. 161–176, Springer 2002.

[11] S. Ciraci, W. Havinga, M. Aksit, C. Bockisch, and P. van den Broek. “A graph-based aspect interference detection approach for UML-based aspect-oriented models”. In Transactions on Aspect-Oriented Software Development VII, pp. 321–374, 2010.

[12] T. Dinkelaker, M. Mezini, and C. Bockisch. “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.

[13] V. Hourdin, J. Y. Tigli, S. Lavirotte, G. Rey, and M. Riveill. “SLCA, composite services for ubiquitous computing”. In Proceding of the 5th International Conference on Mobile Technology Applications and Systems(Mobility), 2008.

[14] M. Issarny, N. Caporuscio, and Georgantas. “A perspective on the future of middleware-based software engineering”. In Future of Software Engineering. FOSE'07. pp. 244-258. IEEE, 2007.

 ISBN: 978-1-61208-171-7