MULTIMODALITY AND CONTEXT-AWARE ADAPTATION

Quentin Limbourg and Jean Vanderdonckt
Université catholique de Louvain (UCL), School of Management (IAG), ISYS-BCHI
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
Phone: +32-10 / 478525 – Fax: +32-10 / 478324
E-mail: {limbourg, vanderdonckt}@isys.ucl.ac.be
Web: http://www.isys.ucl.ac.be/bchi/members/{qli,jva}

Abstract: In principle, context-aware adaptation is assumed to bring to the end user the benefit of adapting the user interface currently being used according to significant changes of the context of use in which the user interface is manipulated. To address major shortcomings of system that hardcode the adaptation logic into the user interface or the interactive software, a mechanism is introduced to express context-aware adaptation as a set of logical production rules. These rules are gathered in graph grammars and applied on graphs representing elements subject to change and conditions imposed on the context of use. These rules can express both adaptations within the same modality of interaction (*intra-modality adaptation*) and across several modalities of interaction (*trans-modality adaptation*).

Keywords: Adaptation, Context of use, Graph grammars, Graph transformations, Intra-modality adaptation, Production rules, Trans-modality adaptation.

1. INTRODUCTION

The context of use is typically considered as a potential source of information to trigger an adaptation of the User Interface (UI) of a system according to significant changes of some properties of interest (Thevenin, 2001). The context of use is hereby defined as a triplet (*U*,*P*,*E*) where *U* represents the user and her properties (e.g., demographics attributes, skills, preferences, native language, motivations), *P* represents the computing platform and related properties (e.g., screen resolution, interaction capabilities, devices), and *E* represents the environment in which the user is carrying out the interactive

task on the computing platform (Calvary *et al.*, 2003). Similarly, the environment is described by attributes like organisational structure, psychological parameters (e.g., level of stress). Any change of the current value of any of the *U*, *P*, and *E* parameters can potentially indicate a change of the context of use. However, in practice, only some of them truly represent a significant change of the context of use that should have an impact on the user interface. The adaptation logic that reacts to these significant changes of context is generally embedded in the software (i.e. hardcoded), thus resulting into little or no flexibility for changing it. In addition, the adaptation logic is rarely expressed in a formal way that is immediately executable by an automaton without requiring further modification. To address these shortcomings and to enable any person to express an adaptation rule according to the same language that can be communicated, we relied on the mechanism of graph transformations (Freund *et al.*, 1992) that is further explained in the next section. The steps of the methodology are (Limbourg & Vanderdonckt., 2004):

1. The context of use is represented by a graph (with nodes and arcs).
2. Other models that are typical of model-based approaches for multi-platform UIs (Paternò & Santoro, 2002) (e.g., presentation and dialog of the UI) are also represented by graphs.
3. The adaptation logic is expressed by transformation rules that check existing graphs for satisfying conditions of applicability and apply them consequently so as to create new specifications imposed on the adapted UI that can then be rendered.
4. The adaptation logic can be executed statically at design time or dynamically at execution time (Kawai *et al.*, 1996).

2. GRAPH TRANSFORMATION FOR CONTEXT-AWARE ADAPTATION

TOMATO consists of a general-purpose methodology that systematically applies design knowledge to produce a final UI by performing different steps based on a transformational approach. This approach enables expressing and simultaneously executing transformation of models describing UIs viewpoints. Fig. 1 illustrates the transformations steps supported in TOMATO:

- **Reification** is a transformation of a high-level requirement into a form that is appropriate for low-level analysis or design.
- **Abstraction** is a transformation of a low level the extraction of high-level requirement from a set of low-level requirements artefacts or from code (Bouillon *et al.* 2003).
- **Translation** is a transformation a UI in consequence of a context of use

change.

- **Reflection** is a transformation of the artefacts of any level onto artefacts of the same level of abstraction, but different constructs or various contents.
- **Code generation** is a process of transforming a concrete UI model into a compilable or interpretable code.
- **Code reverse engineering** is the inverse process of code generation.

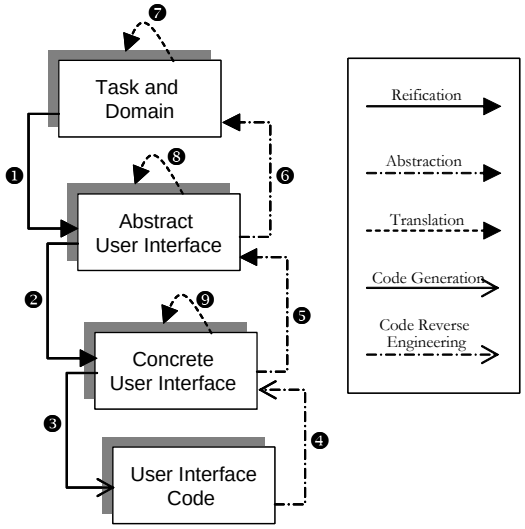

Figure 1. Transformations between viewpoints.

The different transformation types are instantiated by *development steps* (each occurrence of a numbered arrow in Fig. 1). These development steps may be combined to form *development paths*. While code generation and code reverse engineering are supported by specific techniques, we use graph transformations to perform model-to-model transformations i.e., reifications, abstractions and translations. TOMATO models have been designed with an underlying graph structure. Consequently any graph transformation rule can be applied to any TOMATO specification. Graph transformations have been shown convenient formalism for our present purpose in (Limbourg *et al.*, 2004). The main reasons of this choice are (1) an attractive graphical syntax (2) a clear execution semantic (3) an inherent declarativeness of this formalism. Development steps are realized with transformation systems. A transformation system is a set of (individual) transformation rules. A transformation rule is a graph rewriting rule equipped with negative application conditions and attribute conditions (Roszenberg, 1997).

Fig. 2 illustrates how a transformation system applies to a TOMATO specification: let *G* be a TOMATO specification, when 1) a Left Hand Side (LHS) matches into *G* and 2) a Negative Application Condition (NAC) does not matches into *G* (note that several NAC may be associated with a rule) 3)

the LHS is replaced by a Right Hand Side (RHS). *G* is consequently transformed into *G'*, a resultant TOMATO specification. All elements of G not covered by the match are considered as unchanged. All elements contained in the LHS and not contained in the RHS are considered as deleted (i.e., rules have destructive power). To add to the expressive power of transformation rules, variables may be associated to attributes within a LHS. Theses variables are initialized in the LHS, their value can be used to assign an attribute in the expression of the RHS (e.g., LHS : button.name:=x, RHS : task.name:=x). An expression may also be defined to compare a variable declared in the LHS with a constant or with another variable. This mechanism is called 'attribute condition'.

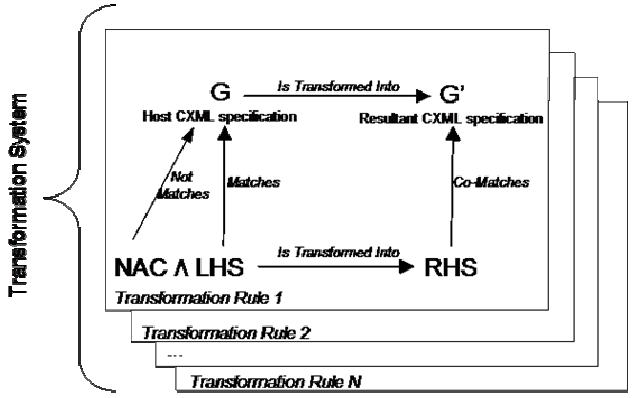

Figure 2. A transformation system in TOMATO methodology.

3. ADAPTATION TO CONTEXT CHANGE

According to the Cameleon reference framework (Calvary *et al.*, 2003), adaptation with respect to the context change can take place at three levels (Fig. 3): (1) at the "task & domain" level where one or both models are affected to reflect a change of context of use (e.g., a change in the organisational structure may move a task from one role to another one, thus resulting in deleting this task from the task set of a person); (2) at the "abstract UI" level, where the UI is described independently of any modality of interaction; (3) at the "concrete UI" level, where the UI is described with specific modalities, but still independently of any computing platform. In terms of graph transformations, context adaptation covers model transformations adapting a viewpoint to another context of use. This adaptation is performed at any of the three above levels.

Fig. 4 depicts a production rule that perform the following adaptation: for each pair of abstract individual component mapped onto concurrent tasks, transfer all facets of the abstract individual component that is mapped onto the task that is target of the concurrency relationship, to the other abstract

individual component. Abstract individual components represent a sort of abstraction of interaction objects independently of their modality of interaction. As such, they are located higher than traditional Abstract Interaction Objects (Vanderdonckt & Bodart, 1993). This rule should not be applied to task that still have decomposition. In other words, the rule is applied only on leaf tasks of the task model (Paternò & Santoro, 2002).

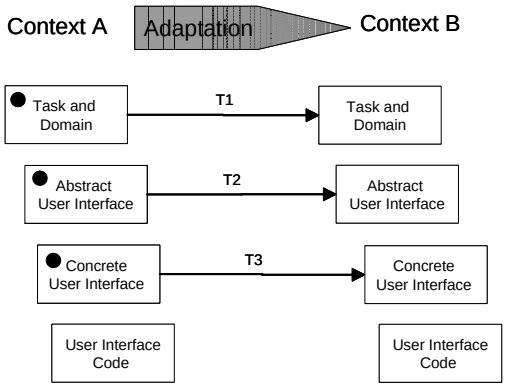

Figure 3. Context adaptation at different levels of TOMATO.

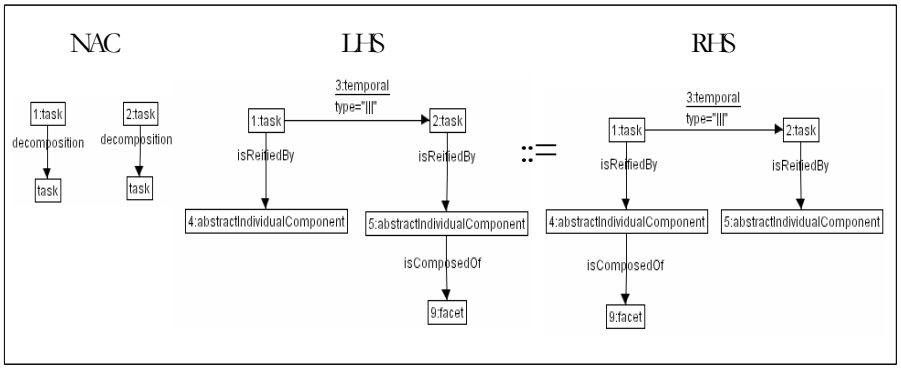

Figure 4. A merging of facets of abstract individual components

ACKNOWLEDGEMENTS

We gratefully acknowledge the support of the European IST CAMELEON research project (http://giove.cnuce.cnr.it/cameleon.html), the CAMELEON partners for fruitful exchanges and discussions, and the SIMILAR network of excellence (http://www.similar.cc), the European research task force creating human-machine interfaces similar to human-human communication of the European Sixth Framework Programme.

REFERENCES

Bouillon, L., Vanderdonckt, J., and Chow, K.C.: 2004, Flexible Re-engineering of Web Sites. In: *Proceedings of 8th ACM Int. Conf. on Intelligent User Interfaces (IUI'2004)*. Funchal, 13-16 January 2004. ACM Press, New York, pp. 132-139.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J.: June 2003, A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers* **15**(3) 289–308.

Freund, R., Haberstroh, B., and Stary, C.: 1992, Applying Graph Grammars for Task-Oriented User Interface Development. In: Koczkodaj, W.W., Lauer, P.E., Toptsis, A.A. (eds.): *Proceedings of 4th International Conference on Computing and Information (ICCI'92)*. Toronto, May 28-30, 1992. IEEE Computer Society Press, Los Alamitos, pp. 389–392.

Kawai, S., Aida, H., Saito, T.: 1996, Designing Interface Toolkit with Dynamic Selectable Modality. In: *Proceedings of 2nd ACM International Conference on Assistive Technologies (ASSETS'96)*. Vancouver, April 11-12, 1996. ACM Press, New York, pp. 72–79.

Limbourg, Q. and Vanderdonckt, J.: 2004, Transformational Development of User Interfaces with Graph Transformations. In: *Proceedings of 5th International Conference on Computer-Aided Design of User Interfaces (CADUI'2004)*. Madeira, January 14-16, 2004. Kluwer Academics Publishers, Dordrecht.

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., and Trevisan, D.: 25 May 2004, USIXML: A User Interface Description Language for Context-Sensitive User Interfaces. In: *Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages" (UIXML'04)*

Paternò, F. and Santoro, C.: 2002, One Model, Many Interfaces. Chapter 13. In: Kolski, Ch., Vanderdonckt, J. (eds.), *Proceedings of 4th International Conference on Computer-Aided Design of User Interfaces (CADUI'2002)*. Valenciennes, May 15-17, 2002. Kluwer Academics Publishers, Dordrecht, pp. 143–154.

Rozenberg, G. (Ed.): 1997, *Handbook of Graph Grammars and Computing by Graph Transformation*, Volume 1: Foundations, World Scientific.

Thevenin, D.: December 2001, *Adaptation en Interaction Homme-Machine: le cas de la Plasticité*, Ph.D. thesis. Grenoble, France, 2001. On-line: http://iihm.imag.fr/publs/2001

Vanderdonckt, J. and Bodart, F.: 1993, Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In: Ashlund, S., Mullet, K. Henderson, A., Hollnagel, E., White, T. (eds.): *Proceedings of the ACM Conference on Human Factors in Computing Systems (InterCHI'93)*. Amsterdam, April 14-19, 1993. ACM Press, New York, pp. 424–429.