Automated planning for User Interface Composition

Yoann Gabillon Mathieu Petit Gaëlle Calvary Humbert Fiorino

University of Grenoble, CNRS, LIG
385, avenue de la Bibliothèque, 38400, Saint-Martin d'Hères, France
{yoann.gabillon, mathieu.petit, gaelle.calvary, humbert.fiorino}@imag.fr

ABSTRACT

In ubiquitous computing, both the context of use and the users' needs may change dynamically with users' mobility and with the availability of interaction resources. In such changing environment, an interactive system must be dynamically composable according to the user need and to the current context of use. This article elicits the degrees of freedom User Interfaces (UI) composition faces to, and investigates automated planning to compose UIs without relying on a predefined task model. The composition process considers a set of ergonomic criterions, the current context of use, and the user need as inputs of a planning problem. The user need is specified by the end-user (e.g., get medical assistance). The system composes a UI in turn by assembling fragments of models along a planning process.

Categories and Subject Descriptors

H.5.2 [User Interfaces]: Ergonomics, Graphical user interfaces (GUI), Prototyping, User-centered design. D2.2 [Software Engineering]: Design Tools and Techniques, User-Interfaces.

General Terms

Design, Human factors, Algorithms.

Keywords

User Interfaces composition, Semantic models, Automated task planning, Context of use.

1. INTRODUCTION

Pushed forward by new information technologies, Weiser's vision of ubiquitous computing comes to reality [11]. His definition of ambient computing implies 1) a global knowledge of an information system context, and 2) adaptation processes to comply with a given context of use. The context of use is usually defined as a <user, platform, environment> triplet. Unpredictable contexts of use might affect users' interactive behaviors and task organization. Therefore, each User Interface (UI) design option from the task model to the final UI is highly contextual and might be decided at runtime. Therefore, most of the ubiquitous design frameworks consider variations of the context of use as inputs to select UI options (i.e., plastic design [9], automatic generation [6], mashups [1]). However, to the best of our knowledge, the user task variation is usually left out.

This article outlines an approach, based on automated planning, to support task as well as UI variations in an integrated framework for UI composition. In the following, section 2 exemplifies multi-level UI composition on a medical support case study. Section 3 elicits the degrees of freedom UI composition faces to. Section 4 introduces automated planning and highlights the UI composition process. Section 5 presents an integrative framework for UI composition by planning. The focus is set on the composition of models (*Model-based composer*) and code (*Code composer*). Section 6 summarizes our contributions and draws some perspectives.

2. RUNNING CASE STUDY

Victor is a New-York citizen on vacation in Philadelphia. After spending his day tasting the rich local food, Victor feels bloated at night and needs to find the doctor on duty. Using his PDA, he specifies his need in general terms: "I would like to get medical support".

According to Victor's need and to the available interaction resources and existing information, the system *abstracts* the goal, *plans* a task model, and *composes* one possible UI. The composition process is not fully autonomous: it requires additional information from Victor. The negotiation UIs (Figure 1) are composed by the system as well.

Given Victor's current location, the system asks Victor whether he prefers to return home or to find assistance in Philadelphia (Figure 1a). Victor chooses to consult a local doctor. The system therefore finds and provides him with possible local contact information: the nearest hospital or doctor on duty, a medical hot-line, or the firemen (Figure 1b).

(a) Possible locations. *(b) Possible options.*

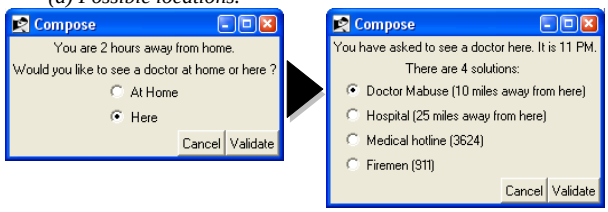

Fig. 1. Automatically composed UI.

Victor selects the doctor on duty. The systems provides him with contact and location information. The UI layout matches the current user platform:

Smartphone. If Victor prefers to keep information at hand, a UI is generated for his Smartphone. With respect to the limited screen resolution, pieces of information are tabbed and no additional data is provided (Figure 2).

SEMAIS'11, Feb 13 2011, Palo Alto, CA, USA

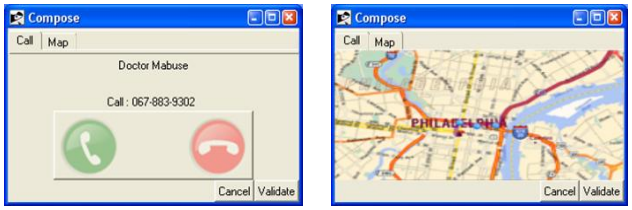

Fig. 2. The generated UIs for a Smartphone.

Desktop Wall. If a desktop wall is available, the system generates a single pane UI allowing to contact and/or to get route information to the doctor's office. Additional information about close services, like the nearest all-night chemist, is also provided (Figure 3).

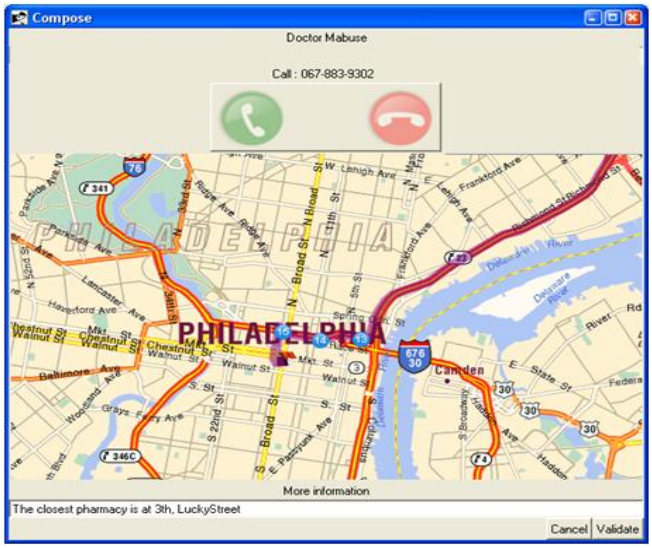

Fig. 3. The UI generated for a desktop wall display.

3. MODELS ARE KEY

This section goes back to model based design in Human Computer Interaction (HCI), and claims for keeping these models at runtime so that to support dynamic adaptation.

3.1 Model based design

UIs are modeled along several levels of abstraction. For example, the CAMELEON reference framework identifies four main levels of design decisions [2]. The *task model* (TM) describes how a given user task can be carried out; the *abstract UI* (AUI) delineates task-grouping structures (i.e., workspaces); the *concrete UI* (CUI) selects and layouts the interaction elements (i.e., interactors) into the workspaces; at last, the *final UI* (FUI) is about the code. Mappings relate these models to each other. For example, a task should be mapped to one workspace of the AUI at least.

In a dynamic context of use, any of these UI design decisions and their subsequent models and mappings might be updated at runtime to match the current context of use. As long as these adaptations satisfy the usability and utility properties, the UI is said to be *plastic* [9]. In Victor's case study, every design decision might be adapted in a plastic way. For example, the task "Find nearest chemist" may be removed from the task model. The AUI model associated to the Smartphone favors the "Call the office" subtask whilst the desktop wall version gives a simultaneous access to the two subtasks ("Call the office" and "Find route information"). Variations at the CUI level are not exemplified in the case study. We could imagine a switch from a route display to a list of directions so that to fit with the Smartphone display. Such adaptations might be seen as a transformation between two graphs of models.

3.2 Graph of models to support adaptation

Earlier work defined principles for UI plasticity [8]. The authors structured the CAMELEON reference framework as a network of models and mappings (Figure 4), and claimed for keeping this graph alive at runtime so that to support adaptation.

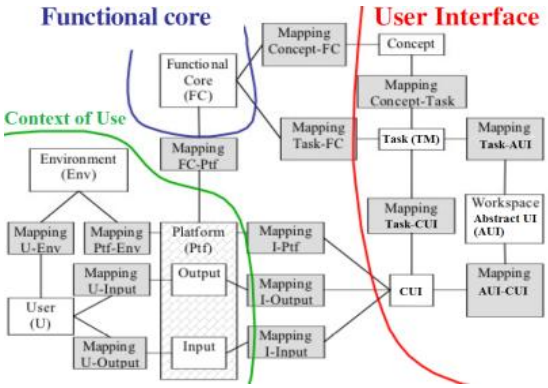

Fig. 4. Semantic graph of models of an interactive system [8].

The graph expresses and maintains multiple perspectives on a system. For example, a UI may include a task model, a concept model, an AUI model and a CUI model linked by mappings. In turn, the UI components are mapped onto items of the Functional Core, whereas the CUI interactors are mapped onto the input and output (I/O) devices of the platform. Although such a model provides a helpful organizational view on the elements and relationships involved when designing a plastic interactive software, the proposed mappings between the context of use and the other components hardly describe contextual choices inside each model (TM, CUI, AUI, etc.).

Demeure et.al. provide a complementary semantic graph of models to control UI plasticity within each design option level [4]. Their model allows UI designers to check out replaceable (i.e. functionally equivalent) units at run-time. For example, a given layout of interactors at the CUI level might be switched to another one depending on the desired ergonomic properties [7]. We propose to replace these hand-made choices by predicates dependent of the context of use, and manipulated by the system.

Figure 5 illustrates the design process along the models and mappings proposed in [8] and the replaceable options described in [4]. For example, at the task level (TM), two options exist for T2 depending on the context of use (Figure 5 b&c).

In Figure 5, within a level of abstraction, units relate to each other according to a *consumer-provider* relationship (Figure 5: $c \mapsto p$ link). For example, at the TM level, one of the options for the task T2 relies on the occurrence of a provider leaf option[1] for the task T3 (Figure 5a). Therefore, as T2 "consumes" T3, this option will be triggered if and only if T3 is satisfied. Depending on the current context of use, *consumer-provider* links behave like

[2]A *leaf option* has no relationship for neither providing nor reifying options.

"opened" or "closed" transistors. In a given $c \mapsto p$ relationship, the status of a transistor depends on the contextual requirements of the provider (p). For example, at the TM level in Figure 5, one of the task T2 options is possible only for experienced users (Figure 5d).

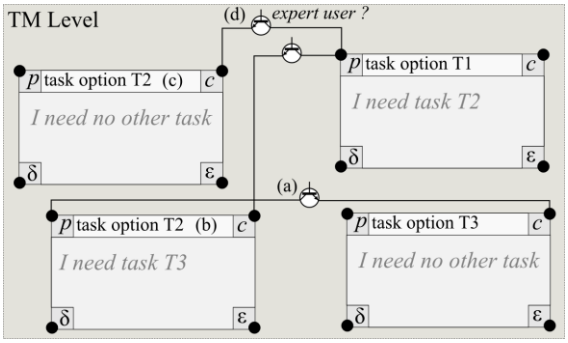

Fig. 5. Example of a TM options graph.

In UI design, mappings link together options of different levels of abstraction. For example, interactors from the CUI level are usually mapped onto workspaces of the AUI level. These mappings, presented in Figure 4, or the definitional links in [4] constitute *abstracting-reifying* relationships between the options of distinct CAMELEON levels of abstraction (Figure 6: $\delta \mapsto \varepsilon$ links).

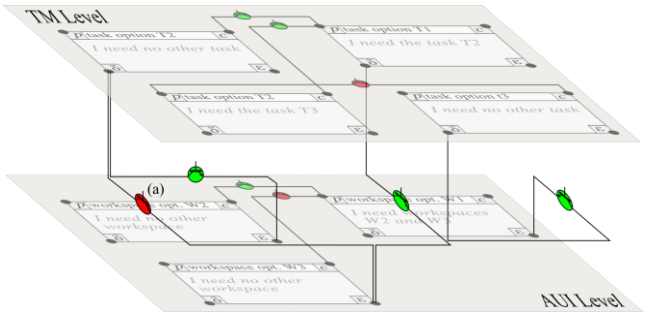

Fig.6. Abstracting-reifying relationships between two design options at the TM and AUI levels of abstraction.

For example, the TM level presented in Figure 5 might be reified into several options of an AUI level (Figure 6). In Figure 6, a task option T1 is reified into a workspace layout "W3" of the AUI level. Like the $c \mapsto p$ relationship, $\delta \mapsto \varepsilon$ relationship between levels of abstraction makes sense in a given context of use only. For example, Figure 6 depicts a runtime configuration where the workspace layout W3 cannot reify the task T2 given the current context of use (Figure 6 a).

The relationships we propose ($\delta \mapsto \varepsilon$ and $c \mapsto p$) for modeling software can easily be explored automatically. The next section investigates automated planning.

4. UI COMPOSITION BY PLANNING

This section presents the core principles of planning and shows how this approach is valuable for UI composition.

4.1 Principles of automated planning

An automated planning algorithm derives a temporal sequence of *actions* into a *plan* to accomplish a given *goal* [5]. For example, in the previous case study, the sequence {"Call the doctor"→"Find route information"} is a plan made of two actions. A Planning algorithm pipes syntactic processes to perform symbolic computations. Such logical reasoning is formally described by a finite-state machine where actions are transitions between possible *states of the world*. Actions are defined by sets of pre/post-conditions. Pre-conditions specify the run-time dependencies of an action while post-conditions are met after executing the action. For example, Victor's Smartphone should be connected *(pre-condition)* to display a location map *(action)*. When this action is executed, the map is eventually displayed *(post-condition)* on the Smartphone. An updated state of the world integrates these new post-conditions, therefore enabling further actions.

4.2 Automated planning for UI composition

A planning solver algorithm computes a transition graph between an initial state of the world and a final state corresponding to the system/user goal. Currently, such algorithms are mainly applied to service composition [10]. However, as illustrated in our case study, context-dependent UI composition and automated planning strongly relate. Thus, we propose to address UI composition by planning where:

- "Actions" are "User interfaces options". Existing components (e.g., the UI associated to the task "Call the office") are actions for the planner;
- The "State of the world" is made of the current "Context of use" and the "Ergonomic properties" to be satisfied. For example, the fact *"Victor owns a Smartphone"* is a predicate of the state of the world;
- The "selected plan" is the "composed UI". For example, the UI displayed on the Smartphone is a concretization of the plan {"Choose the city"→"Choose the doctor" →"Contact the doctor"→{"Call the office"→"Find the route information"→ "Find the nearest pharmacy"}} computed by the planner.

Even if several challenges still need to be worked out to bridge the gap between automated planning and UI composition, next section presents "Compose", a first framework for rapidly prototyping UIs by planning. Its use by end-users belongs to the future.

5. THE COMPOSE FRAMEWORK

Compose is a proof of concept of UI composition by planning. It has been built on top of several functional Java-coded components (Figure 7).

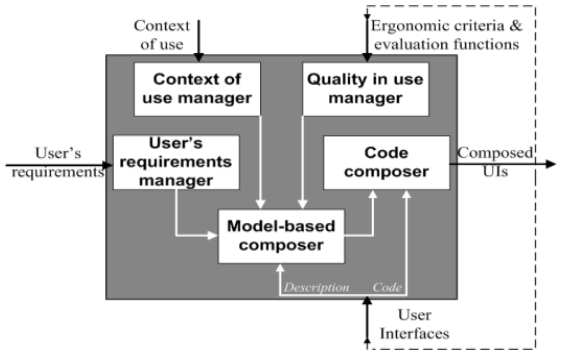

Fig. 7. Functional decomposition of Compose.

The *Context of use and quality in use managers* translate the required ergonomic criteria and the current context of use into

predicates. These assertions define the current state of the world. For example, the predicate *Has("User","Desktop Wall")* is true when Victor stands nearby a managed desktop wall.

The *User requirements manager* expresses a user need as a goal to be met. For example, Victor's need would be to "Get medical support".

The *Model-based composer* and the *code composer* are the core components of Compose. The model-based composer handles the planning process, whilst the code composer translates a resulting plan into a FUI. In the current prototype, planning is applied to the task level only. Once the TM level is composed, mappings are made with a generic purpose graphic toolkit called COMET [3]. COMETs are reusable context-aware widgets defined at the task level and reified along the CAMELEON reference framework. The next sections focuses on the core components of Compose.

5.1 Model-based composer

The model based composer takes actions as inputs and structures them into a plan. This planning process is twofold: at first, the user task modeling is composed by collating predefined subtasks (Figure 8(p1)); next, each task (i.e.: the planner actions) is mapped onto a UI (Figure 8(p2)). These selections bring out a composed UI (i.e., the selected plan) whose properties match the current state of the world. The resulting plan is a semantic description of the UI to be composed.

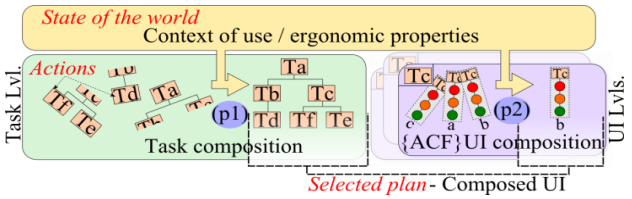

Fig. 8. Compose planner instantiation.

In Victors' case study, Compose waits for a user need specification (i.e. "Get medical support"). The composer tries to find a corresponding TM level entry point. The option "Get Medical Support" is selected. The planning algorithm then explores the semantic network of $c \mapsto p$ relationships between the task options of the TM level (Figure 9). For each uncovered task option, Compose checks whether it is possible or not to map the task onto a COMET and render the UI. These mappings are derived according to the current state of the world. For example, leaf task options like "Choose the city" or "Choose the doctor" might be mapped onto a UI as soon as Victor's platform is available whatever the characteristics of the platform are (in Figure 9: t1 & t2). Other task options like "Call the office" rely on carrier capabilities at the platform level (in Figure 9: t3). "Contacting the doctor" option distinguishes between several screen sizes and resolutions (Figure 9: t4 & t5). When a large screen is available, such a sub-task option involves tree leaf options (Figure 9: u1), while on a Smartphone display, solely two of them are displayed (Figure 9: u2).

Once all contextual pre-requisites of a provider option are met, the relationships to his consumers turn green and each of them might in turn be checked-out. After a provider/consumer relationship status has been specified, the state of the world is updated with the new facts the providing option concurs to establish. For example, when "Choose the city" pre-requisites are met, the composer knows for sure that Victor will be able to specify his searching location and the fact "The location has been set" is added to the state of the world.

Figure 9 outlines the status of the $c \mapsto p$ relationship between the task options after Compose has explored and checked-out a state of the world wherein Victor interacts on a desktop wall display.

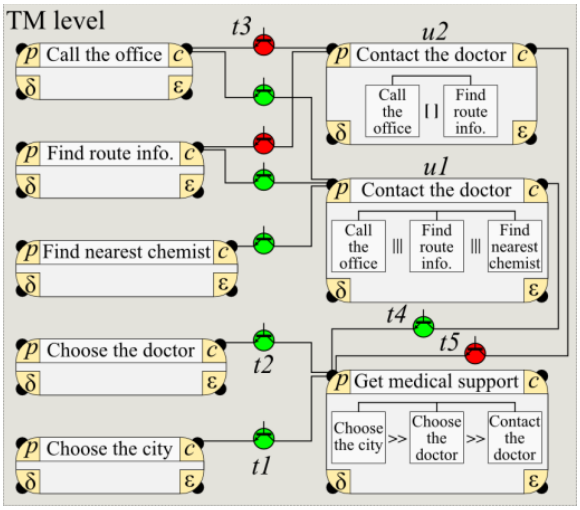

Fig. 9. Possible TM level planning when a desktop wall is available.

Such contextualized semantic UI model highlights the appropriate task factorization in a given context of use. When a green path of provider-consumer relationship is established from the provided objective to the leafs task options, a task tree has been found to achieve the user goal. In such case, the code composer is provided with the planned task tree. Subsequent mappings are made between tasks and COMETs to derive the final UI.

5.2 Code composer

The *Code composer* derives the UI code from the graph of models at the task level. At design time, the options of the task level have been statically associated to COMETS. Therefore, in Compose, each action of the plan is reified by a contextualized COMET. For example, the option "Get medical support" is mapped to a COMET laying out a sequence of frames on the desktop wall. The *Code Composer* brings these pieces of UI together in a unified layout. For instance, the desktop wall task tree provided by the model-based composed is mapped to the COMET presented in Figure 10. For example, the action "Get medical assistance" is mapped to a "COMET C7" laying out a sequence of frames on the desktop wall. These frames contain several sub-COMETs ("COMET {C3, C1, C4}") to map the task options "Choose the city", "Choose the doctor" and "Contact the doctor". In turn, the mapping "COMET C4", that reifies the task "Contact the doctor", contains several vertically aligned sub-COMETs. These sub-COMETs ("COMET {C2, C5, C6}") are mapped in the same way.

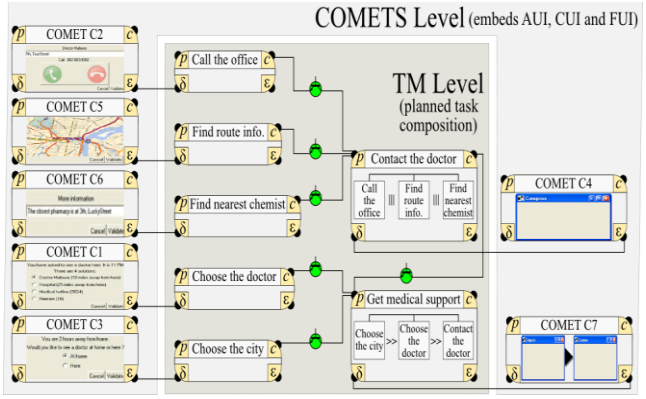

Fig. 10. The "Desktop Wall" planned task tree. Each task is reified by a pre-defined COMET.

6. CONCLUSION AND FUTURE WORK

This article outlines a work in progress to support opportunistic user needs. A UI is composed by selecting a path in a graph of models according to the current context of use and the ergonomic properties to be satisfied. UI composition is seen as a planning problem. So far, the focus has been set on the model-based composer whatever the *time* is: design time for the designer thus providing a rapid prototyping tool, or runtime for the end-user as an intelligent assistant.

Future works include improvements of planners to fully support UI composition. This means (1) generating trees (i.e., tasks structures) instead of sequences, (2) defining appropriate functional and implementational software architectures for general-purpose ubiquitous computing, (3) taking non functional properties into account (i.e., returning the best plan instead of the first one). Thus, beyond perspectives in HCI, this work has challenged planning for ubiquitous computing.

7. ACKNOWLEDGMENTS

This work has been mainly founded by the "Informatique, Signal, Logiciel Embarqué" research cluster of the Rhône-Alpes region. It has also been supported by the french "ANR MyCitizSpace" and the european ITEA2 UsiXML projects.

8. REFERENCES

[1] Brodt, A., Nicklas, D., Sathish, S., and Mitschang, B. 2008. Context-aware mashups for mobile devices. In *WISE 2008: Web Information Systems Engineering*. Springer-Verlag, 280-291.

[2] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. 2003. A unifying reference framework for multi-target user interfaces. *Interacting with Comp. 15*, 3, 289-308.

[3] Demeure, A., Calvary, G., and Coninx. 2008. K. COMET(s), A Software Architecture Style and an Interactors Toolkit for Plastic User Interfaces. In *15th Int. Work. on Interactive Systems Design, Specification, and Verification*. Springer-Verlag, 2008, 225-237.

[4] Demeure, A., Calvary, G., Coutaz, J. and Vanderdonckt, J. 2006. The COMETS Inspector: towards run time plasticity control based on a semantic network. *In Proceedings of the 5th Int. Workshop on Task Models and Diagrams for User Interface Design: TAMODIA'06*, Springer LNCS 4385, Haselt, Belgium, 324-339.

[5] Nau, D., Ghallab, M., and Traverso. P. 2004. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.

[6] Paternò, F., Mancini, C., and Meniconi, S. 1997. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. *Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction*, Chapman & Hall, Ltd., 362-369.

[7] Scapin, D.L. and Bastien, J.M.C. 1997. Ergonomic criteria for evaluating the ergonomic quality of interactive systems. *Behaviour & Information Technology*. Colchester, ROYAUME-UNI: Taylor & Franci *16*, 4 (1997), 220-231.

[8] Sottet, J-S., Ganneau, V., Calvary, G., Coutaz, J., Demeure, A., Favre, J-M. and Demumieux, 2007. R. Model-driven adaptation for plastic user interfaces. *In Proc. of the 11th IFIP TC.13 Int. Conf. on Human-Computer Interaction : INTERACT'07*, Springer LNCS 4662, Rio de Janeiro, Brazil, 397-410.

[9] Thevenin, D. and Coutaz, J. 1999. Plasticity of user interfaces: Framework and research agenda. *Human-computer Interaction, INTERACT'99: IFIP TC. 13 , 30th August-3rd September 1999*, IOS Press, 110.

[10] Traverso, P. and Pistore, M. 2004. Automated Composition of Semantic Web Services into Executable Processes. *Proceedings of ISWC, LNCS*, 380-394.

[11] Weiser, M. 1991. The computer for the 21st century. *Special Issue on Communications, Computers, and Networks 272*, 3, 78-89.