A version of this paper has been published in:
Communications of SIWN, Vol. 3, June 2008, pp. 46-52
(formerly: System and Information Sciences Notes)

ISSN 1757-4439 (Print)
ISSN 1757-4447 (CD-ROM)

See: http://siwn.org.uk/press/sai/cosiwn0003.htm

Context aware Business adaptation toward User Interface adaptation

Anas HARIRI, Dimitri TABARY, Sophie LEPREUX, Christophe KOLSKI

University of Valenciennes and Hainaut-Cambrésis
LAMIH – CNRS UMR 8530, Le Mont-Houy
F-59313 Valenciennes cedex 9 (France)
Email: {anas.hariri, dimitri.tabary, sophie.lepreux, christophe.kolski} @univ-valenciennes.fr
http://www.univ-valenciennes.fr/LAMIH/

Abstract: The growth of user mobility has created new user needs. Cellular phones, personal assistants and modern communication means are now required for all user tasks. Users want services that are specifically mobile, but they also want to work with and have access to the same applications on their mobile devices as they have at their work place. The mobile system must thus be adapted according to the task and the context (i.e., platform, user and environment characteristics). Usually, adapting an interactive system implies costly User Interface (UI) redesign and reimplementation procedures, with the goal of adapting the business components to the human activity, selecting the interface appropriate to the task and the context. This paper describes an adaptive UI design approach that allows UI to be generated and dynamically adapted to changeable contexts, while preserving the UI's utility and usability. In our approach, patterns are used to connect UI adaptation problems, with their multiple contexts, to usable design solutions. This approach is illustrated by a case study.
Keywords: User interface adaptation, business component adaptation, user interface generation.

1. Introduction

The world is becoming more and more interconnected; new types of technical devices appear nearly every day and become a necessary part of daily life. Traditional user interface (UI) design methods focus on the design stage, after which the generated UI is implanted in the target device. However if the context of use (i.e., general information on the platform (P), the user (U), and the environment (E), as defined in §3.2) changes, the system has to be transferred back to the designer for a redesign stage, during which the system is modified manually to take into account the changes in the context of use. Consider, for example, a company employing hundreds of workers who use different terminal types and generations, who work in different environments, and who can also change their location in the building and work on other machines. Imagine how many system versions must be generated for this company, and how many times the system must be redesigned to take new contexts of use into account. The multitude of devices, with diverse platforms, different user characteristics and expectations, and various work environments, has upped the stakes, creating new desires that must be satisfied [15].

The next section provides a review of the literature. The third section introduces the method proposed for adapting the UI according to various contexts of use, including the business components and patterns used to adapt interactive systems. The section four illustrates the method with a case study. The last section offers our conclusions and proposes several perspectives for future research.

2. Literature Review

Mark Weiser [25] first described what is now called *pervasive computing* [13]. Weiser predicted that intelligent information devices and a variety of technologies would become ubiquitous and that people would be able to use the devices and the technologies at any time and in any place. Schilit and Theimer [22] were among the first to describe an interactive system that was sensitive to context. Their context-aware system used information about the location and the identity of the neighboring users, and tried to take contextual changes into account. Following up on other work in this field, Brown et al. [3] used a wide range of information about the environment and the user's knowledge level in order to provide information and services to the user. Pervasive computing, requiring new methods of human-computer interaction, led to a new generation of interactive systems. The UI in a pervasive computing environment must be able to adapt itself dynamically to its context of use, making it

necessary for the UI to be able to detect information relevant to the current platform, environment and user [6] in order to use them during its adaptation. Other authors call the UI adaptation required by the pervasive environment *UI plasticity* [4]: a plastic UI must be able to intelligently detect and use contextual information (e.g., user location), date and time, proximity of other users, available resources, target platform characteristics, environmental conditions). During runtime, the system must react dynamically to contextual changes in order to respect UI utility and usability criteria. Research on dynamic adaptation has been presented by Mckinley [17], who worked on dynamic modifications of the system's composition and the software's behavior, in response to changes in the execution environment.

Some approaches tend to solve the problem of dynamic modifications of the system's composition. Coninx et al. [5] focus on dynamic UI generation but do not take context changes during runtime or UI adaptation into account. Seffah and his colleagues have proposed multi-user interface design based on task models and patterns [23], using HCI patterns for cross-platform transformations. Javahery et al. [14] have proposed a method for designing UI that takes the context into account, using patterns to make the development process more rigorous. In their work, each pattern is associated with a set of variables that acquires the context of use of a pattern. However, they concentrated only on the design phase, not on UI adaptation during runtime.

The goal of the research presented in this paper was not only to design an interactive system, but also to adapt that system to the context, using business components to alter the functional components and the UI simultaneously.

3. Method based on business components

The proposed method and its constitutive elements are presented in the following sections. This method is part of a global development approach using model-based development.

3.1 The general principles for UI generation

The method executes several activities, which are presented in Fig. 1. In the first stage, the interactive system's business structure is built based on the task analysis. The second stage focuses on the UI design. When the interactive system is being used, a context acquisition is executed to detect any contextual changes. If a change is detected, the utility and the usability of the UI are evaluated [24], and depending on the result of the evaluation, the method returns to one of the previous stages in order to adapt the interactive system. Utility concerns whether or not, in principle, the system function can do what is needed [10], and usability is related to how well users can use that function [18]. If the utility criterion is not respected, the business structure must be modified, and thus the method returns to stage 1. If the usability criterion is not respected, the method returns to the design step, and thus the method returns to stage 2, in order to redesign the UI.

Fig. 2 shows the “build business structure” activity (i.e., stage 1). The pattern library (respectively business components library) is constituted during the design stage using the patterns (respectively business components) extracted from past know-how. Details about the business components are given in section 3.3 and 3.4, and details about the patterns are given in sections 3.5 and 3.6. First of all, a list of patterns is selected from the pattern library. This list is used to facilitate the choice of the business components and their assembly. The patterns chosen depend on (1) the tasks to be performed and (2) the patterns' domain of application. Each of the business components corresponds to one or more tasks. When developing the patterns, the developer determines the knowledge that will facilitate the choice of the business components. Then, respecting the patterns chosen, the business components are selected from a business components library (the business components are classified by domain of application). The selected business components are configured in accordance with the system task specifications (e.g., setting up the search-type property of a “Search” business component based on one of its associated task) and then are connected by a "glue" [2], which is a means of component assembly. This glue is appropriate for the middleware imposed by the software communication interface; this glue is specified and adapted to the needs of the business components' communication interfaces.

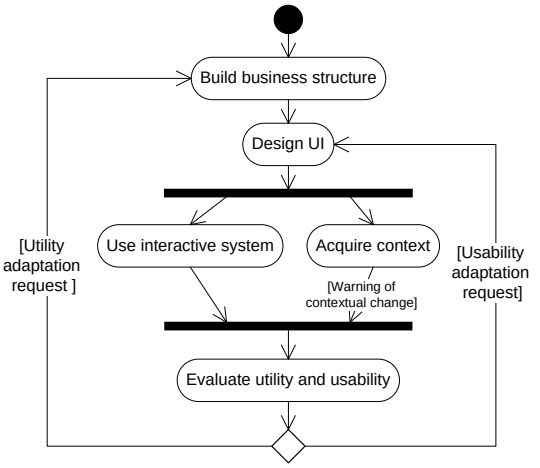


Fig. 1. Activity diagram illustrating the proposed method.

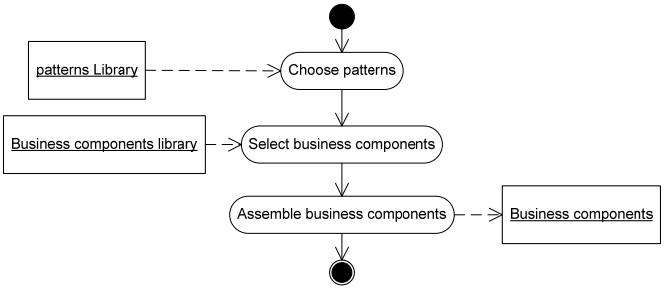


Fig. 2. Focus on the “build business structure” activity.

Fig. 3 shows the “Design UI” activity (i.e., stage 2). Patterns are chosen from the library. The context of use determines the type of the patterns chosen. Patterns are chosen based on their domain of application and the business structure of the system and are used to select presentation components for the business components. In fact, the business component may have several presentation components usable in different contexts. The presentation components that are appropriate for a context of use are chosen using the selected patterns. Then, the presentation components are assembled and configured. The presentation components are arranged on the UI according to a style guide that offers additional information and specifications for the target terminal. Fig. 4 shows extracts from a PDA style guide (left) and from a PC style guide (right). As the figure shows, there are significant differences in screen size and in the icons to be used for system messages.

The code for an initial version of the adaptive interactive system is generated and distributed on the platform. The following section provides more detailed explanations of the context of use and the two types of libraries. (For more details, please see references [11], [12] and [16]).

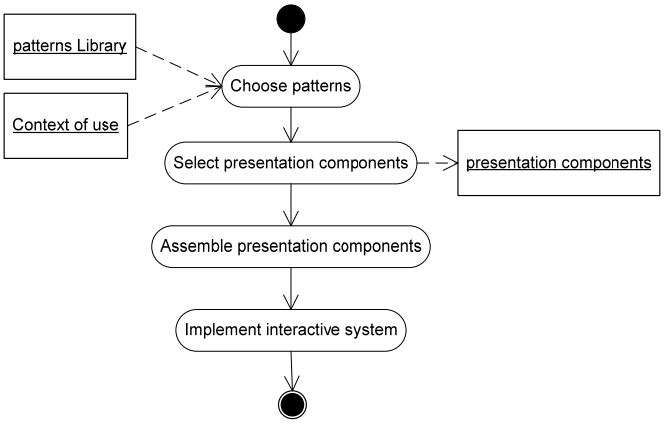


Fig. 3. Focus on the “Design UI” activity.

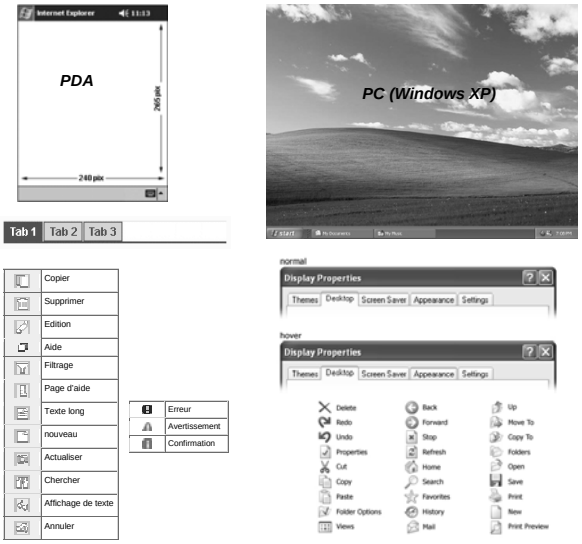


Fig. 4. Example of two style guides.

3.2 Context of use

The context of use must be considered when designing and executing the system. Three categories of contextual information can be distinguished [7]:

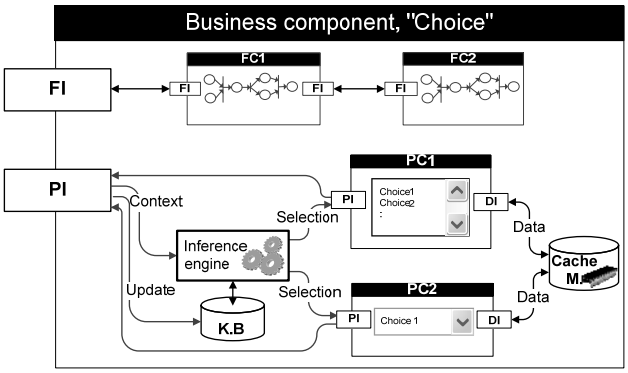
- User information (U), such as the user profile, knowledge level, physical location and/or current activity (e.g., leisure activity, work);
- Platform information (P), such as the terminal characteristics (e.g., processor, memory, interface networks, peripheral devices), connectivity networks, screen size, and/or available interactive tools; and
- Environmental information (E), such as noise level, luminosity, temperature, and/or temporal information (e.g., date, time).

This contextual information is stored and updated during context acquisitions. The captured information is regularly sent to the system so that it can evaluate the utility and usability of the UI. In case of an important contextual change, an adaptation can be requested in order to preserve system utility (business element) and/or usability (UI element). (The modalities of the context acquisition are outside of the scope of this paper. The interested reader can consult references [13] and [19].)

3.3 Business components

Fig. 5 shows an example of the business component, “Choice”. This component is composed of a functional input/output interface, another interface for the presentation components, a knowledge base (KB) used by an inference engine to select presentation components according to the context of use. A cache memory is integrated in the component’s architecture, space in the memory resources used by the presentation components during runtime; in this space, user information and entries can be backed up for transfer and then restored on a new platform, particularly in cases of platform migration (i.e., the system business components must insure a dynamic adaptation without losing the data relating to the current user).

The last business component also has two functional components, and manages two presentation components of type *ComboBox* and *ListBox*, which correspond to two different contexts. During adaptation, the UI part of the business component, “Choice”, can be projected towards *ComboBox* presentation component, and vice versa, depending on the context of use (the KB of the business component and/or the patterns makes the presentation choice decision). This technique helps to avoid retransmitting the UI after each important contextual change.



PC1, PC2: Presentation Component;
FC1, FC2: Functional Component;
FI: Functional Interface;
PI: Presentation Interface;
DI: Data Interface;
KB: Knowledge Base:

Fig. 5. Example of business component.

The KB works to minimize the adaptation processing time during runtime, reduce the use of network connections, and optimize the response time to the contextual changes. This KB must also work in synchronization with the patterns. A minimum KB is integrated into the business component during the design stage, and plays a role in the decider mechanism that chooses the presentation components according to the context of use. The knowledge is integrated in part during the design stage and in part by the machine learning process at runtime. When the system is executed, the Machine Learning (ML) processes improve the knowledge base. (For more details, see reference [19].

Our method uses the C4.5 algorithm [20], a well-known supervised learning classification algorithm used to construct decision trees from examples. The C4.5 uses an information gain ratio criterion, selecting a test that maximizes the value of the gain ratio. Fig. 6 shows a decision tree constructed by C4.5 for the selection of the presentation components based on the platform and, if necessary, the screen size.

```
-support = PC      :    ___ selectPC (listBox)
-support = PDA     :    ___ ScreenSize>=320*250: selectPC (listBox)
                        ___ ScreenSize<320*250: selectPC (comboBox)
-support = PHONE :      ___ selectPC (comboBox)
```

```
<KB id="DT000345"> // Define a knowledge base
 <concept>selectPC</concept> // Indicate the concept (decision goal)
 <rule id="DT000345_R0000001"> // First rule
    <condition>
       <support>PC</support> // Support is PC
    </condition>
    <reaction><selectPC>listBox</selectPC></reaction> // ListBox is selected
 </rule>
 <rule id="DT000345_R0000002"> // Second rule
    <condition>
       <support>PDA</support> // Support is PDA
    </condition>
    <reaction>
       <condition><screenSize> // and screen size is more than 320*250
             <morethan>320*250</morethan>
       </screenSize></condition>
       <reaction><selectPC>listBox</selectPC></reaction> //ListBox is selected
       <condition><screenSize>
             <lessthan>320*250</lessthan> // Screen size is less than 320*250
       </screenSize></condition>
       <reaction><selectPC>comboBox</selectPC></reaction>// ComBox is selected
    </reaction>
 </rule> // Third rule
 <rule id="DT000345_R0000003">
    <condition><support>PHONE</support></condition>// Support is phone
    <reaction><selectPC>comboBox</selectPC></reaction> // ComBox is selected
 </rule>
</KB>
```

Fig. 6. Decision tree for the selection of the presentation components for the business component, "Choice", and its representation in XML.

3.4 The business components library

In order to facilitate the use of the business components, a library contains a list of the business components classified according to their domain of application. This library knows all the available business components and, if they are not available locally, it knows their location. It stores information about each business component (e.g., its goal, its platforms compatibility, and the interfaces used to connect the component). Depending on the designer's preferences, this information can be stored in XML format or another format. Other research (e.g., [9] and [21]) has used formal description, for example.

3.5 Patterns

For Alexander and his colleagues [1], "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice". The patterns have been adapted in software development by Gamma et al. [8]: "Each design pattern lets some aspect of system structure vary independently of other aspects, thereby making a system more robust to a particular kind of change". They proposed a catalogue of design patterns based on template given for a design problem. These design patterns are used also in the HCI domain (see for instance [14] or [21]). Our method proposes the patterns presented in Table 1. Each pattern is actually transcribed in a XML file so as to facilitate the storage and the transmission of the pattern information.

Table 1. Pattern structure.

ID	This field allows each pattern to be identified, by storing the pattern in a base and making it possible to find it easily.
Name	The pattern name must be significant (e.g., useful) and must correspond to the problem that the pattern allows to be solved.
Description	The description expresses the goal of the pattern textually.
Context of use (P, U, E)	The context of use provides information about the context, making it possible to use this pattern.
Launch	The launch gives an idea of the pattern's granularity level. If the pattern contains an executable code, it could be launched automatically. On the contrary, if the pattern contains solutions with a higher granularity level, the pattern will not be able to launch without an interpreter, or sometimes without human intervention.
Solution	This is the solution that is provided to the problem for the given context of use. The solution can be basic (e.g., the solution is a modelling type) or more refined (e.g., the solution is an executable code).
Collaboration	Collaboration makes it possible to call upon another pattern to solve a sub-problem of the main problem.

3.6 The pattern libraries

Our method includes two pattern libraries:

- One pattern library provides patterns that help choose the business components most appropriate to the task model; these patterns can also be used to recommend composition solutions (cf. Fig. 2).

- Another pattern library provides patterns that help choose and assemble the presentation components during the design and adaptation stages (cf. Fig. 3).

4. Case study

The case study scenario involves a user whose goal is to go to the cinema. A Personal Digital Assistant (PDA) is used to search for a film available on the current date based on the actors appearing in the film.

4.1 Build business structure

The pattern library provides one pattern (whose ID number is *PAT10201*) that corresponds to this problem (see. Fig. 7). The pattern focuses on the business component "*Search for a film*" with the search-type "*search by actor*".

```
<pattern> //Defining pattern
   <id>PAT10201</id> // Pattern id
   <name>businessComponentSelection1</name> //Pattern name
   <description>Selection of business components</description> //Description
   <utilizationContext>design</utilizationContext> //it can be used at design time
   <useContext> //Defining the context of use
      <platforms>//It can be used for PC, PDA or cellular phone
         <platform>pc</platform>
         <platform>pda</platform>
         <platform>phone</platform>
      </platforms>
      <environments> //In any environment
         <environment>any</environment>
      </environments>
      <users> //for any user
         <user>any</user>
      </users>
   </useContext>
   <launching>auto</launching>//It can be launched automatically
   <solution> //Describe the solution
      <condition> //Condition: platform: PC& task: search for a film by actor
         <operation type="and">
            <task>searchFilm</task>
            <platform>pc</platform>
            <searchType>actor</searchType>
         </operation>
      </condition>
      <components> //Reaction: selection of a component of type searchFilm
         <component searchType="actor">searchFilm</component>
      </components>
      ... //Repeat for another type of task or platform
   </solution>
   <collaborations> <!-- It collaborates with another pattern for connecting
                           selected components -->
      <patternId>PAT10225</patternId>
   </collaborations>
</pattern>
```

Fig. 7. Example of a pattern.

This *PAT10201* pattern was generated with a tool called PaDev (i.e., Pattern Developer) that allows designers to easily develop their own patterns and to import solutions described in Java or other languages (see Fig. 8).

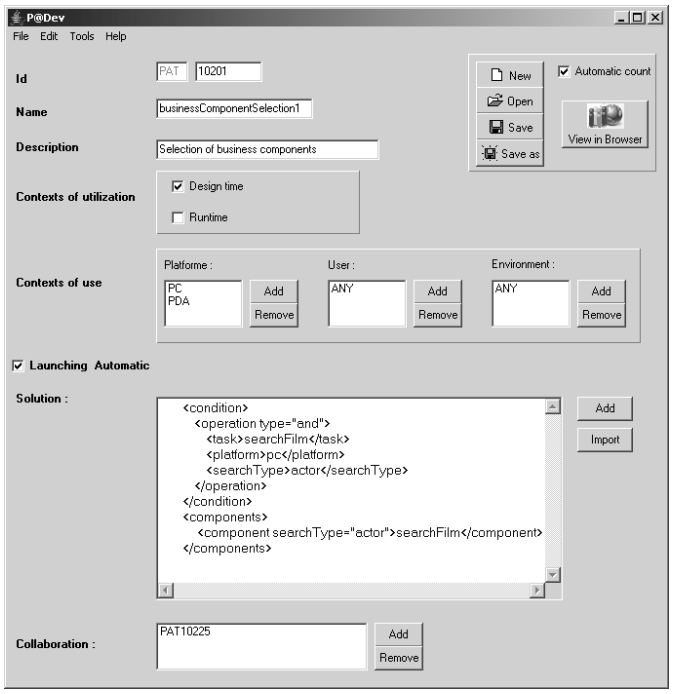


Fig. 8. PaDev: Pattern Development tool

The business component “Search for a film” uses three sub-components (“Navigation”, “Choice” and “Search”) (see Fig. 9). These sub-components are selected from the business components library automatically, so that it is transparent for the user.

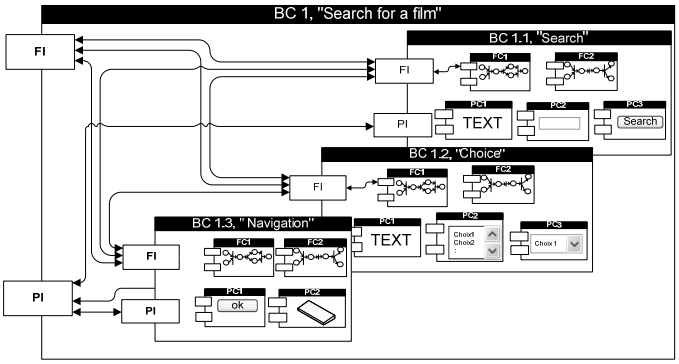


Fig. 9. Business structure of the task “search for a film”

4.2 Design UI

In this case study, the following context of use is assumed:

U={Adult, English, Entertainment},

P={PDA},

E={Alone, Cinema, good weather, 20 pm}.

Patterns, provide by pattern library for presentation component choice, are used to build the presentation aspects. Fig. 10 shows a part of the UI and its links to the business components.

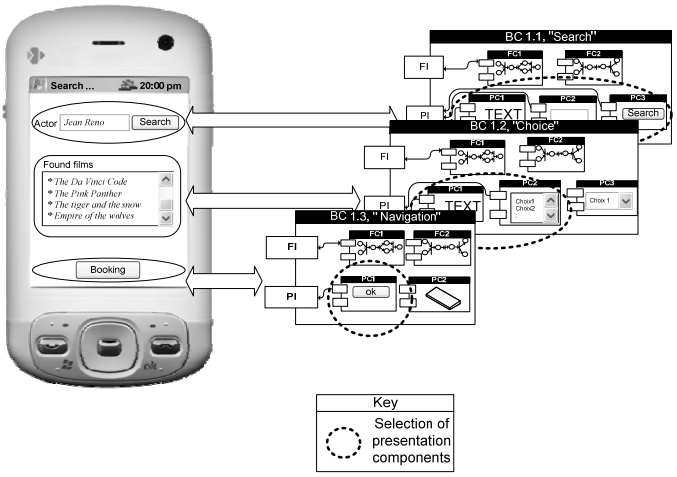


Fig. 10. The UI linked with its presentation components.

4.3 Change in context

The case study scenario changes as follows: while interacting with the PDA, the user receives a phone call from a friend. This friend wants to meet the user so the task changes: instead of searching for a film, the user wishes now to join the friend. The user and environmental context of use is thus changed.

The new context of use is:

U={Adult, English, Personal},
P={PDA},
E={Alone, Street, rainy weather, 20:30 pm}.

4.4 Evaluate utility and usability

When the change in context is detected, the evaluation activity is launched with the goal of insuring that the utility and the usability criteria are respected. In the case study, the context change implies a loss of utility. The user no longer wants the film information, but rather wants the information needed to join the friend. So the evaluation activity sends the method back to the "build business structure" activity in order to adapt the system.

4.5 Adapt business structure

The patterns define the business component needed for the new task: "search the best trip and transport mode" (see Fig. 11). This component requires three components to perform the sub-tasks "prepare the trip", another component to "search the trip" with the transport mode defined and a last component to "guide the user".

The business component related to the "prepare the trip" task contains tree sub-components: two of "Input" type and third of "Navigation" type.

The components work according to the context of use. For example, the "search the trip" component uses the context information to propose a trip appropriate for a young lady alone in the street at 20:30 pm as "search the trip by car".

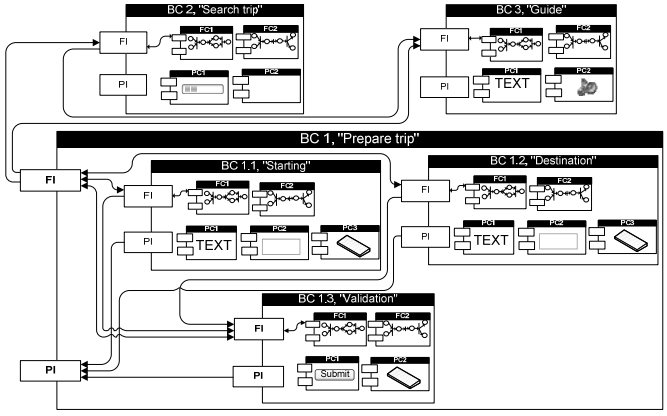


Fig. 11. Business structure of the task "search the best trip and transport mode"

4.6 Adapt UI

At this point, the UI can now be adapted. The context of use allows a graphical interface appropriate for a PDA to be chosen in order to select the starting and destination points (see Fig. 12). The luminosity and contrast are increased because the weather has become rainy. Given the rainy environmental context, the selected presentation component of "guide the user" component is a vocal output interface.

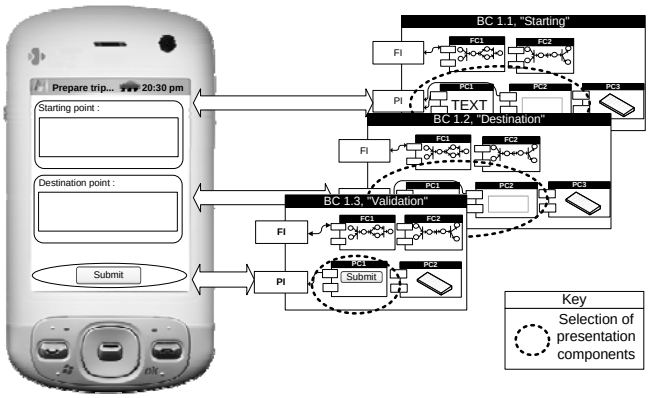


Fig. 12. The new UI linked to its presentation components (for the task "prepare the trip").

5. Conclusion and perspectives

The research presented in this article was designed to develop a method for generating adaptive UI. This method is based on the use of patterns to facilitate the choice of business components related to the system tasks and the presentation components appropriate to the context of use. The case study presented illustrates the use of business components adapt parts of an interactive system, particularly the user interface.

There are many prospects for future research based on the work presented here. For instance, the machine learning system could be studied in more depth. Given too many the patterns, it would be necessary to insure the inter-pattern coherency. Similarly, since the patterns can currently propose several solutions to the same problem in a given context, it would be necessary to refine the reasoning related to the use of these patterns so that the same solution is not chosen systematically. In addition, since the patterns proposed are textually defined in XML files, another possibility would be to use a formalism associated with the definition of ontology to allow the patterns to be directly exploitable.

Acknowledgements

This research was financed by the European Community, the CNRS, the *Nord/Pas-de-Calais* region and the FEDER program.

References

[1] C Alexander, S Ishikawa, M Silverstein, I Fiskdahl-King and S Angel, A pattern language. Oxford University Press, New York, 1977.

[2] B W Beach, Connecting Software Components with Declarative Glue. In Proceedings of ICSE'92, Melbourne, May 1992, ACM Press, pp. 120–137.

[3] P Brown, J Bovey and X Chen, Context-aware applications: from the laboratory to the market place. IEEE Personal Communications, Vol. 2, No. 1, 1997, pp. 1–9.

[4] G Calvary, J Coutaz and D Thevenin, A unifying reference framework for the development of plastic user interfaces. IFIP WG2.7 (13.2) Working Conference, EHCI01, Toronto, Springer Verlag Publ., LNCS 2254, M. Reed Little, L. Nigay Eds 2001, pp. 173–192.

[5] K Coninx, K Luyten, C Vandervelpen, J V den Bergh and B Creemers, Dygimes: Dynamically generating interfaces for mobile computing devices and embedded systems. Proc. 5th International Symposium Mobile HCI 2003, Italy, 2003, Lecture Notes in Computer Science, Human-Computer Interaction with Mobile Devices and Services, L Chittaro, Ed., Springer, Vol. 2795, pp. 256-270.

[6] J Coutaz, J Crowley, S Dobson and D Garlan, Context is key. Communication of the ACM (CACM) 2006, ACM, Ed., pp. 49– 53.

[7] A Dey, Providing architectural support for building contextaware applications. Master's thesis, College of Computing, Georgia Institute of Technology, 2000.

[8] E Gamma, R Helm, R Johnson and J Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
[9] G Godet-Bar, S Dupuy-Chessa and L Nigay, Towards a System of Patterns for the Design of Multimodal Interfaces. In Computer-Aided Design of User Interfaces, Proceedings of 6th International Conference on Computer-Aided Design of User Interfaces CADUI'2006, Bucharest, Springer-Verlag, Berlin, pp. 27–40.
[10] J Grudin, Utility and usability: research issues and development contexts. Interacting With Computers, 4 (2), 209-217, 1992.
[11] M A Hariri, J Petersen, D Idoughi, C Kolski and D Tabary, Plastic presentation of control data in context-awareness environment. Proceedings of EAM'07 European Annual Conference on Human Decision Making and Manual Control, 2007.
[12] M A Hariri, D Tabary and C Kolski, Plastic HCI generation from its abstract model. Proc. of IASTED-HCI Int. Conf. on Human-Computer Interaction Anaheim, USA, 2005, Hamza M.h., Ed., ACTA Press, pp. 246–251.
[13] K Henricksen and J Indulska, Software engineering framework for context-aware pervasive computing. Proc. International Conf. Pervasive Computing and communications Florida, USA, 2004.
[14] H Javahery, D Sinnig, A Seffah, P Forbrig and T Radhakrishnan, Pattern-based UI design: adding rigor with user and context variables. Proceedings TAMODIA 2006, LNCS, Vol. 4385, pp. 97-108.
[15] T Kindberg and A Fox, System software for ubiquitous computing. IEEE Pervasive Computing, Vol. 1, No. 1, 2002, pp. 70–81.
[16] S Lepreux, M A Hariri, J Rouillard, J Tarby, D. Tabary and C Kolski, Towards multimodal user interfaces composition based on usixml and MBD principles. Human-Computer Interaction, Part III, HCII 2007. Lecture Notes in Computer Science (LNCS) 4552 2007, J J., Ed., Springer-Verlag, pp. 134–143.
[17] P McKinley, S Sadjadi, E Kasten and B Cheng, Composing adaptive software. IEEE Computer, Vol. 37, No. 7, 2004, pp. 56– 64.
[18] J Nielsen, Usability engineering, Boston, Academic Press, 1993.
[19] J Pascoe, Adding generic contextual capabilities to wearable computers. Proceedings of 2nd International Symposium on Wearable Computers 1998.
[20] J R Quinlan, C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, California, 1993.
[21] F Radeke, P Forbrig, A Seffah, D Sinnig, PIM tool: support for pattern-driven and model-based UI development. Proceedings TAMODIA 2006, LNCS, Vol. 4385, pp. 82-96.
[22] B Schilit and M Theimer, Disseminating active map information to mobile hosts. IEEE Network, Vol. 8, No. 5, 1994, pp. 22–32.
[23] A Seffah, P Forbrig and H Javahery, Multi-devices « Multiple » user interfaces: development models and research opportunities. Journal of Systems and Software, Vol. 73, No. 2, 2004, pp. 287-300.
[24] A Seffah, M Donyaee, R B Kline and H K Padda, Usability measurement and metrics: A consolidated model. Software Quality Control, Vol. 14, No. 2, 2006, pp. 159-178.
[25] M Weiser, The computer for the 21st century. Scientific American, Vol. 265, No. 3, 1991, pp. 94–104.