Received October 5, 2017, accepted December 2, 2017, date of publication December 13, 2017, date of current version March 9, 2018.
Digital Object Identifier 10.1109/ACCESS.2017.2782880

A UI-DSPL Approach for the Development of Context-Adaptable User Interfaces

THOURAYA SBOUI[1], MOUNIR BEN AYED[2], AND ADEL M. ALIMI[1]
[1]REGIM Laboratory, National Engineering School of Sfax, Sfax 3038, Tunisia
[2]REGIM Laboratory. Faculty of Science of Sfax, Sfax 3038, Tunisia
Corresponding author: Thouraya Sboui (sboui.thouraya@gmail.com)

ABSTRACT Unlike adaptive interfaces which use sensors to adapt themselves, adaptable interfaces need the intervention of end users to adapt their different aspects according to user requirements. These requirements are commonly expressed according to the context of use. This latter was defined by the triplet <platform, environment, user> where the platform refers to the physical device and the device software, the environment refers to the physical environment in which the application is used and the user element refers to the user preferences and user profile. In this paper, we define a dynamic software product line (DSPL) approach for the development of a family of context-adaptable user interfaces. The DSPL paradigm exploits the knowledge acquired in software product line engineering to develop systems that can be context-aware, or runtime adaptable. Our approach satisfies a set of contributions which will be validated by implementing and evaluating them according to an illustrative case study.

INDEX TERMS Context-adaptation, DSPL approach, UI adaptation.

I. INTRODUCTION
To make user interfaces (UIs) more usable and more efficient, it is essential to adapt these interfaces to the context of use change. In order to develop a family of context-adaptable user interfaces, some proposals [4], [6], [23], [35], [36] have opted to use the Model Based User Interface Development (MBUID) paradigm. This paradigm defines a set of models [17] to generate multiple interfaces for different context of use. Other proposals [3], [5], [12], [14], [19], [24], [26], [29], [30] have opted to use the Software Product Line Engineering (SPLE) paradigm [17], [18]. The SPLE paradigm consists of a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way. Furthermore and unlike SPLE, the Dynamic Software Product Line (DSPL) paradigm [7], [16] continues to configure and adapt at the runtime. The DSPL paradigm exploits the knowledge acquired in SPLE to develop systems that can be context-aware or runtime adaptable.

In this context, the present paper describes a UI-DSPL approach dedicated for the development of a family of context-adaptable user interfaces. Our approach (CAUI) includes two phases, a design phase for the development of initial interfaces and a runtime phase for the interface adaptation.

The paper is organized as follows; Section 2 presents related works and their gaps. Section 3 describes the contributions of the CAUI approach. Section 4 highlights the illustrative example. Section 5 presents the design phase and its implementation. Section 6 presents the runtime phase and its implementation. Section 7 presents an evaluation of the proposed approach and a discussion of obtained results. Section 8 presents the related works and Section 9 presents the conclusion and a presentation of perspectives.

II. RELATED WORKS
This section presents an overview of existing UI-SPL approaches. The comparative analysis (table 1) is based on the following criteria:

- **Approach Type**: specifies the proposed approach type. It can be a conventional SPL approach, a dynamic SPL approach, or a model driven software product line approach (MD-SPL approach);
- **Approach concepts**: specifies the concepts used to implement the SPL approach. Among these concepts, is component, aspect, model or any other concepts (e.g. document [19], [20]);

TABLE 1. Comparative analysis of existing UI-SPL approaches. the dimensions were ranked regarding their support. "+" fully explicitly supported, "–" when it is not supported or there is no information.

Related Works	Approach							Context of use			Adaptation		
	Type			Concepts									
	SPL	DSPL	MDE	Component	Aspect	Model	Other	User	Environment	Platform	Design Time	Runtime	Adaptation Model
Schlee & Vanderdonckt, 2004 [30]	+						+	-	-	-	-	-	-
Garcés *et al.* 2007 [13]	+		+			+		-	-	-	-	-	-
Quinton *et al.*, 2011 [31]	+					+		-	-	-	-	-	-
Muller, 2011 [24]	+		+			+		-	-	-	-	-	-
Boucher *et al.*, 2012 [5]	+		+			+		-	-	-	-	-	-
Pleuss *et al.*, 2012 [27]	+							-	-	-	+	-	-
Pleuss *et al.*, 2013 [26]			+			+		-	-	-	+	-	-
Arboleda *et al.*, 2013 [3]	+		+			+		-	-	-	-	-	-
Logre *et al.*, 2014 [22]	+				+	+		-	-	-	-	-	-
Kramer, 2014 [19 ,20]		+					+	-	-	+	+	+	-
Gabillon *et al.*, 2015 [12]		+		+				-	-	+	+	+	-
Sottet *et al.*, 2015 [34]	+					+		-	-	-	-	-	-
Sboui et al. 2017 [33]		+	+			+		+	-	-	+	+	+

- **The context of use**: it denotes what dimensions of the context of use are supported. The context of use is a triplet: platform, user and environment;
- **Adaptation Time**: specifies the type of the supported adaptation. Does adaptation was supported by the design time phase, the runtime phase or by both phases;
- **Adaptation model:** does the approach propose an adaptation model?

Schlee and Vanderdonckt [30] automatically generates the C++ code of a MS Windows user interface that can be adapted at design time by (un)selecting features subject to adaptation from a feature diagram. The designer is responsible for deciding which features, e.g., a command, a button, an icon, should be incorporated in the adapted UI. Therefore, there is no other way for taking the context of use into account, which may result into Meyer seven specification sins: noise, silence, contradiction, sur-specification, etc.

Garcés *et al.* [14] propose a semi-automatic Model Driven Software Product Line approach (MD-SPL). MDA concepts are combined with SPL concepts in order to develop a graphical user interfaces (GUIs). The defined approach is a layered approach; each layer is related to a specific domain (e.g. business, architectural or technological). For each domain, the authors define the metamodel, the correspondent model and the feature model. To move from one level to another, the approach levels are connected by means of transformations. The major defect of [14] is that the developed interfaces are not context-adaptable and are only generated for the Java platform.

Quinton *et al.* [29] propose an automatic software product line approach that generates UIs for mobile devices by merging the feature model (FM) assets. To bridge the gap between application feature diagram (FD) and the device FD, authors propose a pruning process which creates a reduced application metamodel. The role of this metamodel is to check if the product being derived can be executed in a given hardware. The Quinton approach mainly generates mobile devices, furthermore, there is neither context management nor interface adaptation.

Gabillon *et al.* [12] combines MBUID and SPL concepts to develop the graphical user interfaces. In his proposal, Müller put the focus on the layout (disposition of widgets in the container) design. The Müller approach is too theoretical, furthermore, it don't deal neither with interface adaptation nor with context sensitivity.

Boucher *et al.* [5] mention that the direct configuration of FMs is not suitable and apply a concern separation between FMs and UI configurations. To generate the feature model, Boucher used the UI configuration views, the feature configuration workflow and the property sheet. After the FM generation, the features are implemented using the AUI model. The CUI model is generated from the AUI. Boucher *et al.* [5] don't generate the final UIs, they only present the interface sketches. Furthermore, there is neither context management nor interface adaptation.

Pleuss *et al.* [26] propose an approach which includes only a design phase in which the target context element was the customer. Pleuss *et al.* [27] used the Model-Based User Interface Design (MBUID) models [17] to support their approach: a task model representing what the end-user wants to achieve, a domain model representing the data manipulated by the tasks, an Abstract User Interface (AUI) model, and a Concrete User Interface (CUI) model to develop a family of customized UIs. Pleuss *et al.* [27] also used MBUID models to implement the domain and the application engineering levels of a UI-SPL process. In Pleuss approaches, the context was considered only at the design phase. At this phase, the interface was customized/adapted according to the interface customer during its configuration.

Arboleda *et al.* [3] use a model driven approach based on a decision model to derivate a specific product. The decision model takes as input the transformation model (defining the relationship between the feature model, the domain concepts metamodel and the architecture metamodel) and the feature model. The decision model is used with the product model and feature model configuration to generate the final product. The Arboleda approach is a generic approach which is not dedicated nor for UI development neither for UI adaptation. The UI case study was used just to validate the approach.

Logre *et al.* [22] propose an SPL approach for the development of a family of dashboards. In their approach, the authors propose a metamodel which defines dashboards concepts. The meta-model will serve to generate the feature model. This later is implemented using aspects. The presented prototype implements the link between the metamodel and the feature model and provides a semi-automated support for the approach. In [22], there is nor context management neither UI adaptation.

Kramer [19] and Kramer *et al.* [20] uses a DSPL process to develop a platform-adaptive UI. The context of use was considered at the design phase and at the runtime phase. To implement UI variability, the author had used GUI documents as source elements for initiating the process. Kramer deals only with sensed context and adapts the generated interfaces according to the device characteristics.

Gabillon *et al.* [12] propose an automatic Dynamic Software Product Line (DSPL) process that generates a UI able to adapt its behavior when the context changes during the runtime. To generate an adaptive UI, authors used the configured feature model, the current context of use and components for features implementation. Like Kramer, Gabillon deals only with sensed context. Furthermore, he adapts the generated interface at both phases according to the screen size.

Sottet *et al.* [34] define an MD-SPL approach to manage UI variability. The authors define multiple FMs, allowing the separation of concerns and propose a partial and a staged configuration process. Sottet *et al.* [34] do not deal nor with the context of use neither with the interface adaptation.

In addition to UI-SPL approaches, we list in the following other works which deal with interface adaptation. Among these works, we find those [4], [6], [27], [35], and [36] which use the MDE/MBUID paradigm to generate a family of adaptive interfaces and others which propose different techniques to adapt the interface according to the context of use [8], [13].

Calvary *et al.* [6] propose an approach which covers both the design time and run time phases. The Calvary approach (named Cameleon Rreference Framework (CRF)) has now become widely accepted in the HCI Engineering community as a reference for structuring and classifying model-based development processes of UIs that support multiple contexts of use. Calvary deals with a particular interface adaptation which is UI plasticity. This latter is defined by Calvary as ''the capacity of user interfaces to adapt to the context of use while preserving human values''.

Sottet *et al.* [35] propose a model driven approach to generate adaptable UI to the context of use. The authors define models presenting the interface (the task model, the AUI model and the CUI model) and models presenting the context of use. As context elements, the authors target the user, the platform and the environment but practically, they implement their approach according to the platform element.

Bouchelligua *et al.* [4] propose a MBUID approach for the development of plastic interfaces. This approach allows the adaptation to the context of use (platform, environment and user) based on parameterized transformation. In her Approach, Bouchelligua adapts the interface only at the design phase. At runtime, the author doesn't deal nor with context management neither with UI adaptation.

Mezhoudi [23] uses the user feedbacks and the machine learning to adapt her interface. By using the machine learning technique, the final user will have the choice between several adapted interfaces and to him to choose the interface that goes to him. Mezhoudi approach is among the first MBUID approaches that have used user feedbacks to adapt user interfaces.

UsiXml [36] supports a Model Driven Engineering (MDE) approach and covers all CRF models. UsiXml adaptations are focused on the platform model. Users are supported through stereotypes, however there is nor context management neither users involvement during interface adaptation.

Gajos *et al.* [13] propose a system which performs dynamically interface adaptation. The approach targets as context elements, the devices, tasks, preferences and abilities. In their approach, the authors propose an algorithm which finds in less than one second the optimal adapted UI in the solution space. The major defect of Gajos approach is that is not dedicated for the development of a family of UIs.

Cerny *et al.* [8] propose a technique that aims to reduce the development and maintenance efforts of CUI to a level comparable with a single UI. Unlike most of the existing CUI approaches, their technique does not involve an external UI model. Instead, it aims to reflect runtime-information and structures already captured in the application, while extending them to provide an appropriate CUI.

Based on the above analysis, we can list the following shortcomings:

S1) Only 2 out of 12 UI-SPL approaches use the DSPL paradigm to develop a family of user interfaces. These approaches implement a dynamic adaptation, while the rest deal with a static adaptation;

S2) Models are used in MDE and MD-SPL approaches, however they are not yet used in Dynamic SPL approaches;

S3) No UI-SPL approach consider the user while adapting user interfaces. The user was considered only in [8] and [23] approaches.

S4) Based on these shortcomings, we define the following questions, to which our approach have to respond.

A. RESEARCH QUESTIONS

Based on the gaps of existing works, we formulate the following research questions:

RQ1) How to ensure the generation of a family of interfaces while dynamically adapting the interface?

RQ2) How to ensure the abstraction and the reusability of a UI-DSPL approach?

RQ3) How to consider the user while adapting the UI?

RQ4) How to facilitate the design and the development of a runtime adaptation mechanism in a UI-DSPL approach?

To response to these questions, we present in the next section our approach as well as its contributions.

III. THE CAUI APPROACH: A UI-DSPL APPROACH FOR THE GENERATION OF CONTEXT-ADAPTABLE UIs

In this paper, we propose a semi-automatic UI-DSPL approach [33] which generates a context-adaptable user interfaces. The proposed approach includes two phases; a design phase dedicated for the development of initial UIs and a runtime phase for the adaptation of the main UI according to the context change.

As context information, our approach use profiled context information, in particular, the user preferences. Despite the many attempts to automate the acquisition of user preferences, end users are still the first supplier of this kind of data.

In the following, we present the fundamental contributions of our approach.

1) **Manage profiled context information.** Our approach deals with a profiled context at both phases. At the design phase, and in order to generate the context interface, SPL experts will manage the context variability by designing the context feature model, configuring it and generating its interface. While at the runtime phase, the context information will be acquired using the context interface in order to adapt the main interface.
2) **Combine MBUID and SPL concepts.** In order to make our approach more abstract and more reusable, we combine the MBUID and the SPL concepts. At the design phase, the MBUID models (e.g. the Concrete User Interface model, the Final User Interface model) are used to implement the context and the UI variability in order to generate the final interfaces while, they are used at the runtime phase in order to recompose and regenerate the adapted UI.
3) **A runtime adaptation model.** The third contribution of our UI-DSPL approach is the proposition of a design pattern. This latter will facilitate the design and the development of the runtime adaptation mechanism for the designers/developers who want to use our approach to develop a context-adaptable interfaces.

A. THE DESIGN PHASE

The design phase (figure 1) is presented following the Domain Engineering, and Application Engineering processes used in typical SPL process. The domain engineering refers to the design and development of the user interface variability. The Application engineering, on the other hand, refers to UIs derivation, reusing artefacts defined and implemented in the domain engineering level.

Within the domain engineering stage, there are two distinct phases, domain analysis, and domain implementation. In the domain analysis phase, we define the variability models of the application. In order to provide a better separation of concerns, we have defined as variability models two feature models: the context feature model and the UI feature model. This concerns separation ensures the simplicity of design of feature models and the facility of their maintenance.

For the domain implementation phase, it is made up of reusable artefacts and their correspondent code source implementation. At this phase, the feature models (described during the analysis phase) are implemented using the Concrete User Interface (CUI) [17] model. The CUI is the expression of the UI in terms of "concrete interactors" that are modality dependent and implementation technology independent.

For the "Application Engineering" level, it is dedicated for the derivation of specific UIs and it includes two distinct phases: the application analysis phase and the application implementation phase. At the application analysis phase, feature models are configured (i.e. selection of features representing the product to be generated) then at the application implementation phase, we use the model composer that merges features artefacts (corresponding to the selected features). The composed UIs are then transformed, via the

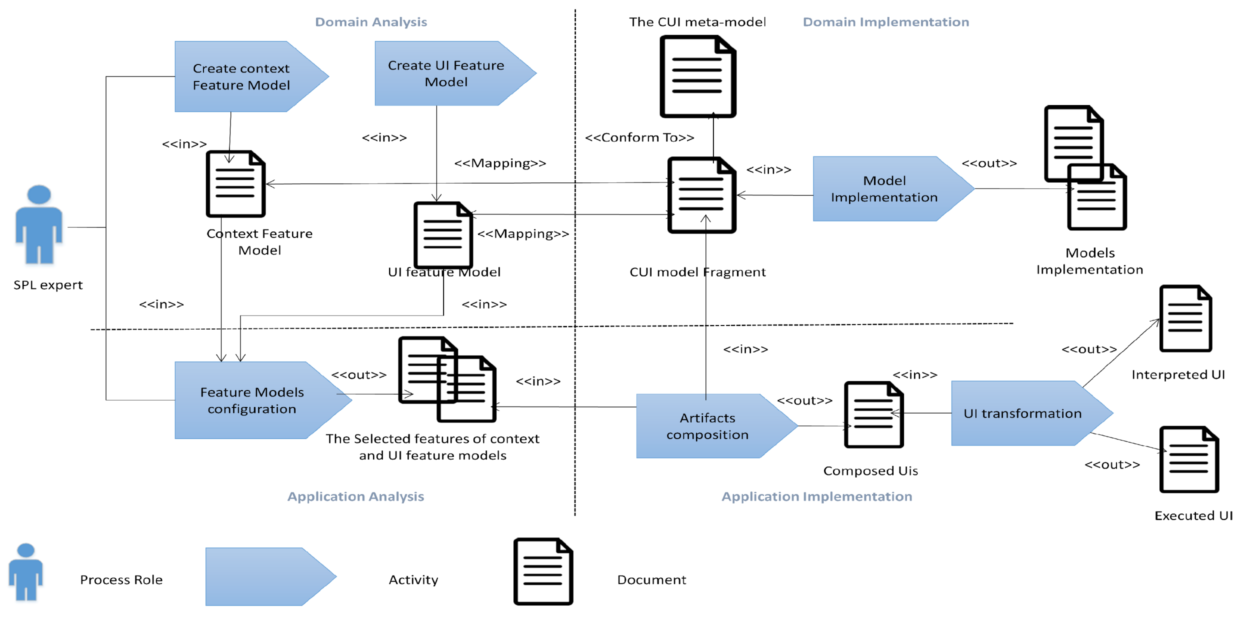


FIGURE 1. SPEM [31] representation of the design phase of the CAUI approach.

code generation techniques, into a Final User Interface (FUI). A FUI [17] is a representation of the UI in any programming language or mark-up language ready for compilation or interpretation.

B. THE RUNTIME PHASE

The runtime phase or the execution phase is the phase which follows the design phase and during which the UI is running. During this phase, the final user can set his preferences. Following preferences settings, an adaptation mechanism is triggered. As showed in figure 2, this mechanism encompasses three main components. The context manager component is responsible for context acquisition and context storage, the adaptation manager is responsible for UI reconfiguration and the recomposition of the new UI and the code source generator is responsible for the generation of the code source of the new UI. This code is ready for interpretation or compilation.

To facilitate the design and the development of the runtime adaptation mechanism, we propose a design pattern. This latter is an adaptation model [32], [33] which describes the main concepts used and implemented by the components of the runtime adaptation mechanism.

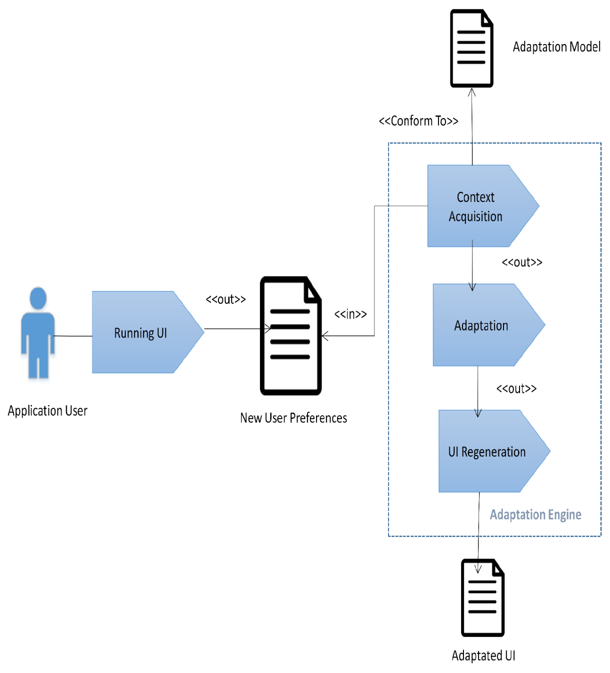


FIGURE 2. SPEM [31] representation of the runtime adaptation mechanism of the CAUI approach.

1) THE RUNTIME ADAPTATION MODEL

As depicted in figure3, the runtime adaptation mechanism is seen as a state machine. States are UI states and transitions are adaptation rules allowing the passage from a source state to a target state.

For UI states (UIstate), the metamodel defines three types: the requiredstate which presents the UI after its adaptation, the defaultstate which describes the UI before its adaptation and the LoadingError state which describes the UI when this later did not succeed to be loaded. A UIstate is defined as a set of aspects (UIaspect) presenting the UI at the current time.

For adaptation rules, they are the constraints defined between features during the design phase. The meta-model defines two types of adaptation rules: the contextconstraint and the aspectconstraint. The contextconstraint rule describes the link between context features and UI features whereas, aspectconstraint describes the link between UI features.

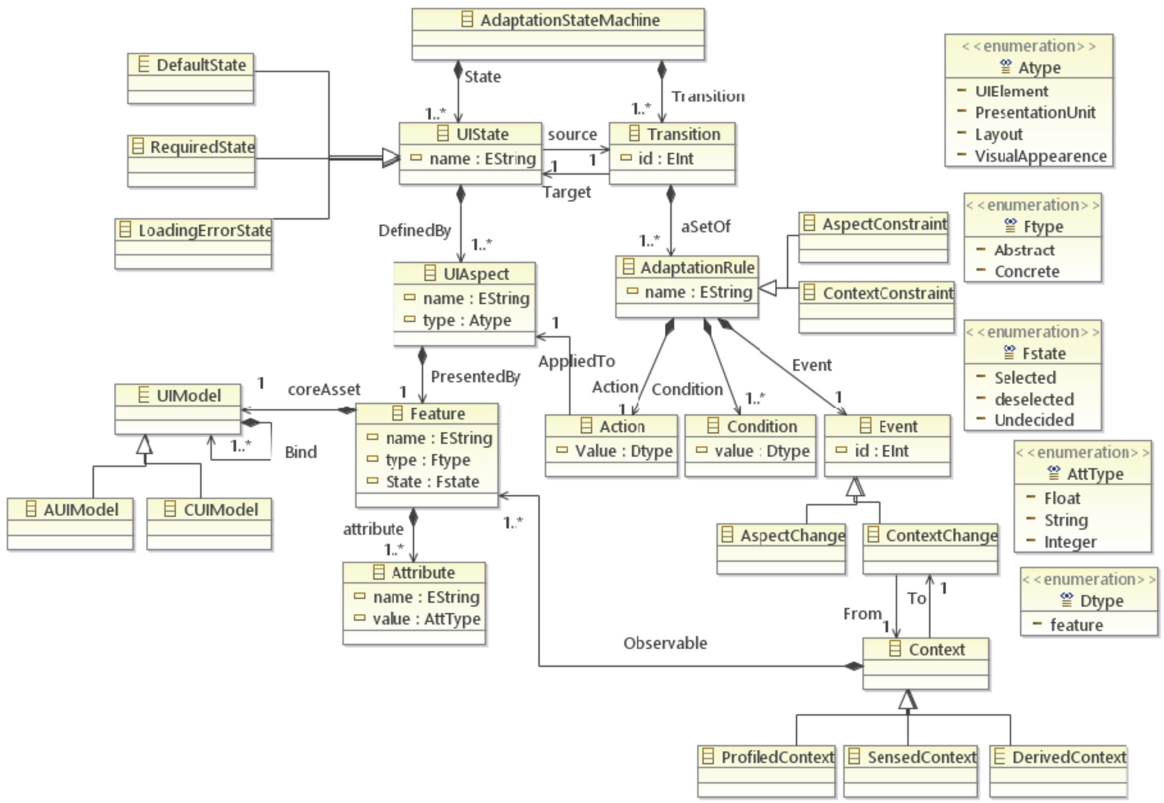


FIGURE 3. The EMF [10] runtime adaptation model.

For Aspectconstraint, only those whom the condition constitute an action of, at least, one Context_rule are considered as an adaptation rule.

Adaptation rule is defined in the form of an Event-Condition-Action (ECA) rule. If adaptation rule is an Aspect-constraint, then the event must be an Aspect_change, else if the adaptation rule is a Contextconstraint, then the event must be a Contextchange.

At the runtime phase, the context and the UI aspect concepts are presented using features and attributes (to indicate feature value).

To rebuild the new UI, the artefacts corresponding to selected features are binded and those corresponding to deselected features are removed. Used artefacts are UImodel. A UI model may be an Abstract User Interface Model (AUImodel) or a Concrete User Interface model (CUImodel).

For enumerative types, the meta-model defines 5 types: 1) atype specify the type of UI aspects 2) ftype specify the type of feature (abstract or concrete) 3) fstate indicates the state of a feature (selected, deselected or undecided), 4) attType indicates the type (float, string, or integer) of the attribute value and 5) Dtype indicates constraint and action values.

To validate the proposed model, we instantiate it according to the following use case. More details about the model and its validation are given in [32].

IV. THE CASE STUDY

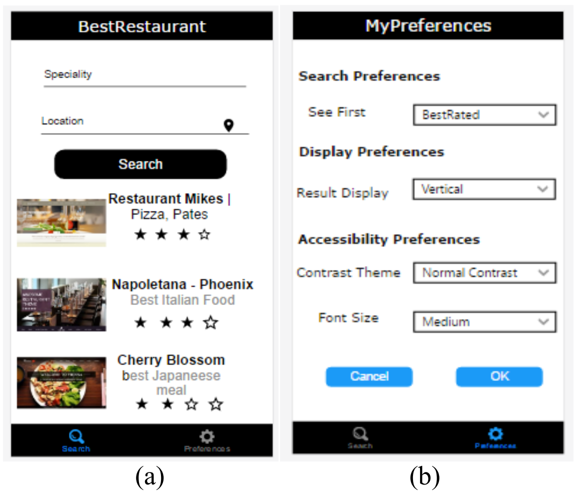
Before presenting the implementation of the design phase and the runtime phase, we describe in this section the case study which will illustrate these phases. The case study highlights the adaptation of the main interface of a ''search for a restaurant'' application to the user preferences change. The user preferences information is a contextual data specific to the end-user. This kind of data may address the customization of two main aspects of a user interface:

- The presentation aspect: customize the UI structure (e.g. containers, widgets), the layout, color, sizing;
- The behavioral aspect: customization of UI element (e.g. the search button) by injecting alternative JavaScript handler.

The illustrative example is about a ''search for restaurant'' application (cf. figure 4). The application has two interfaces: a primary interface for search and a second interface for preferences settings.

The search UI (cf. figure 4 (a)) includes a text field to enter the restaurant specialty, another text field to enter the current location and a search button to validate the search request. By default, the search result is displayed as hyperlink describing the restaurants which correspond to the search request.

The user preferences UI (cf. figure 4(b)) includes three parts. The first part defines the preferences about the



BestRestaurant

Speciality

Location

Search

Restaurant Mikes | Pizza, Pates ★ ★ ★ ☆

Napoletana - Phoenix Best Italian Food ★ ★ ★ ☆

Cherry Blossom best Japaneese meal ★ ★ ☆ ☆

Search Preferences

MyPreferences

Search Preferences

See First: BestRated

Display Preferences

Result Display: Vertical

Accessibility Preferences

Contrast Theme: Normal Contrast

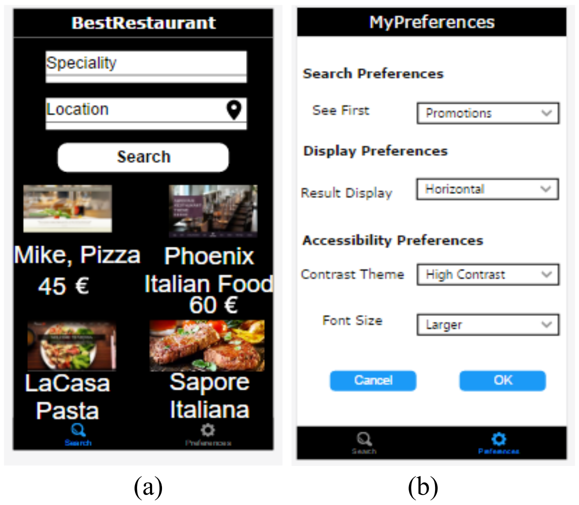
Font Size: Medium

Cancel OK

Search Preferences

(a) (b)

FIGURE 4. Default UIs of the search for restaurant case study. (a) The search UI. (b) The User preference UI.



BestRestaurant

Speciality

Location

Search

Mike, Pizza 45 €

Phoenix Italian Food 60 €

LaCasa Pasta

Sapore Italiana

Search Preferences

MyPreferences

Search Preferences

See First: Promotions

Display Preferences

Result Display: Horizontal

Accessibility Preferences

Contrast Theme: High Contrast

Font Size: Larger

Cancel OK

Search Preferences

(a) (b)

FIGURE 5. The search for restaurant UIs after adaptation. (a) Adapted Search UI. (b) The new preferences Settings.

restaurant the user is looking for, the second part defines user preferences about displaying the search result and the third part is about accessibility preferences. "Search preferences" address the behavioral aspect of the UI, while "display preferences" and "accessibility preferences" address the presentation aspect of the search UI.

To customize the UI behavior, the "Search preferences" include a combobox specifying the type of the restaurant the user is looking for (e.g: best rated restaurant, restaurant offering a promotion). To customize the display of the search result, "Display preference" includes a combobox specifying the display desired by the user (e.g: a vertical display, a horizontal display). While "accessibility preferences" define two comboboxs presenting the criteria used to customize the theme color and the font size of the search UI.

The default User interfaces are described in figure 4. Figure 4 (a) presents the search UI relative to the default user preferences while figure 4 (b) presents the default settings of user preferences. These latter are described as follows: user 1 prefers visualizing the best rated restaurants. The user prefers visualizing the search result displayed vertically in a normal contrast theme and a medium font size. The adapted User Interfaces are described in figure 5. Figure 5 (a) presents the search UI relative to the new user preferences decribed by figure 5 (b). These preferences are described as follows: the user prefers visualizing the restaurants in promotions displayed horizontally in a high contrast theme and a larger font size.

V. THE DESIGN PHASE IMPLEMENTATION

A. DOMAIN ANALYSIS

As described above, the Domain Analysis level includes the definition of the feature models and the constraints defined across their features. Our approach defines two feature models, a context feature model and a UI feature model and a set of feature constraints defined using the ECA rule notation.

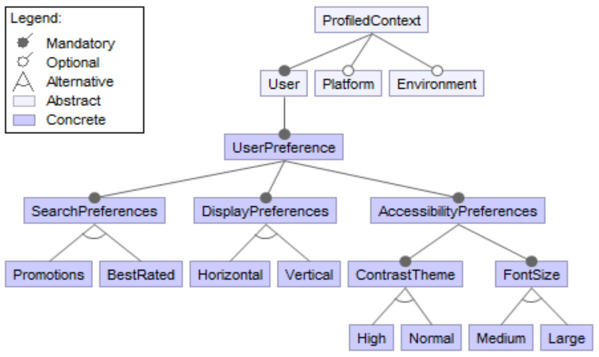


FIGURE 6. The context feature model.

1) FEATURE MODELS

For the context feature model, it describes the context variability. As depicted in figure 6, this variability is expressed across the triplet <user, platform, environment>. UserPreference is a sub-feature of the user feature and defines three variants: SearchPreferences feature, DisplayPreferences feature and AccessibilityPreferences feature. In their turn, the SearchPreference feature defines two variants (Promotions, and bestrated variants), the DisplayPreference feature defines two variants (vertical and horizontal variants) and the accessibilitypreferences feature defines the contrasttheme variant and the fontsize variant. The contrasttheme feature defines two mutually exclusive variants (high and normal variant) while the fontsize feature defines medium and large features, two mutually exclusive variants.

The UI feature model describes the variability of the user interface [25]. In figure 7, the UI variability is expressed across two aspects: the UI structure and the UI presentation. The structure feature defines two variants presenting UI containers (resquestcontainer, and responsecontainer variants).

For the requestcontainer feature, it defines three variants which present the search objects (speciality_textField, location_textfield, and searchButton variants). In turn, the

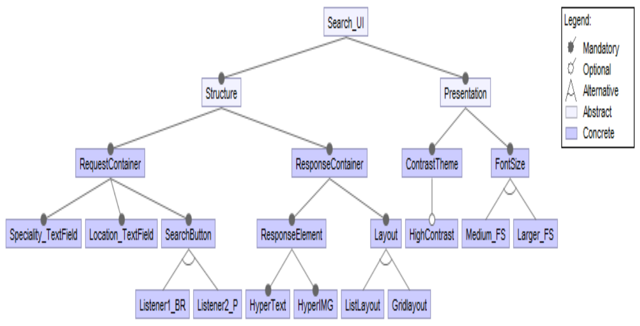


FIGURE 7. The search UI feature model.

searchButton variant defines the listener1_BR variant relative to the listener handling requests of searching the best rated restaurant and the listener2_P variant represents the listener handling requests of searching the restaurants offering promotions.

For the responsecontainer feature, it is defined across two aspects: the UI element aspect (presented by the ResponseElement variant) and the layout aspect (represented by the layout variant). As UI element, the ResponseElement variant presents the response object relative to the search request. A ResponseElement may be a hyperText or a hyperIMG. For the layout aspect, two variants are defined: the GridLayout (i.e. response objects are displayed in a horizontal way) variant and the ListLayout (response objects are displayed in a vertical way) variant.

Regarding the presentation feature, it defines two variants. The contrasttheme variant and the fontsize variant. The former defines one optional feature: the highcontrast feature while the latter defines two mutually exclusive features, the medium_FS and the larger_FS features.

2) FEATURES CONSTRAINTS

In addition to feature models, feature constraints are defined at the domain analysis phase and are describing the link between features of the same feature model (alternatively called intra-model constraint) or between features of different feature models (alternatively called cross-tree or inter-model constraint).

The cross-tree constraints (connecting context features and UI features) express the context sensitivity and are used to adapt statically the interface at the design phase and dynamically at the runtime phase.

The two types of feature constraints (intra-model and inter-model constraints) are defined using the Event-Condition-Action (ECA) language. An ECA rule is described as follows:

On Event (E) **if** Condition (C) **then** Action (A)

Where the event is features presenting the actual context of use, the condition is the connection between context features using the and, or, not operators and the action is the interface features connected using the and, or, not operators.

In the following, we present an example of two feature constraints using the formalism of set theory.

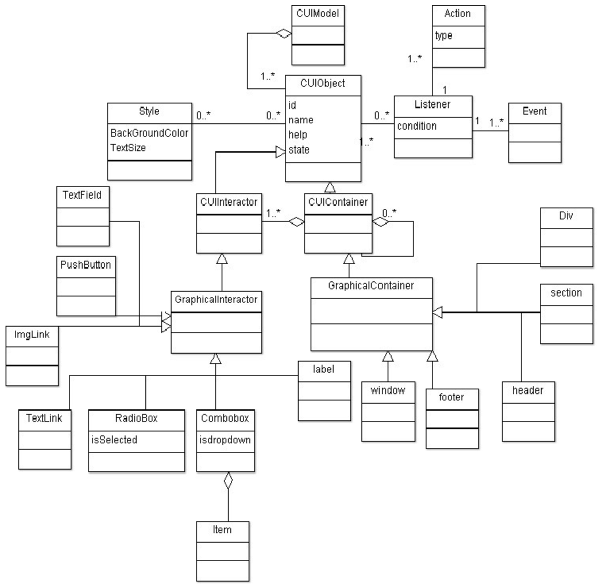


FIGURE 8. Excerpt of the CUI Meta-model [17].

AR1: **on** E = {promotions, horizontal, high, large} **if** C = { horizontal } **then** A = { Gridlayout }, this rule means that the selection of the horizontal feature of the context feature model implies the selection of the Gridlayout feature of the UI feature model.

AR2: **on** E = {BestRated, vertical, normal, medium} **if** C = {BestRated} **then** A = {Listener1_BR}, this rule means that the selection of the BestRated context feature implies the selection of Listener1_BR UI feature (the listener looking for the best rated restaurants).

B. DOMAIN IMPLEMENTATION

As mentioned above, to implement context features and UI features, we used the MBUID models. The used model is the Concrete User Interface (CUI) model. To perform the mapping, we associate for each feature, a fragment of the CUI model. The associated CUI model is an instantiation of the CUI metamodel defined by the UsiXML team [17].

1) THE CUI METAMODEL

Figure 8 presents an excerpt of the CUI meta-model defined in [17]. In addition to the defined classes [17], we define as graphical interactors (graphicalinteractors): the textfield, the pushbutton, the imglink, the textlink, the radiobox, the combobox, and the label classes. Regarding the graphical container (graphicalcontainer), we mainly defined the web containers such as the footer, the header, the section, and the div container. Furthermore and in order to manage the user event (event), we associate a listener class to the CUIobject class. The listener has to implement the Action class in response to the produced event.

2) THE MAPPING BETWEEN FEATURE MODELS AND THE CUI MODEL

In figure 9, we depict an example of mapping between context features and the CUI model. As shown, the search-preferences context feature is mapped into a CUI fragment

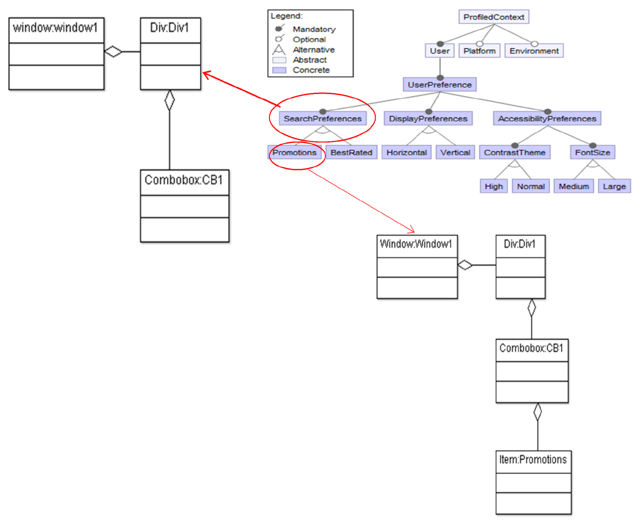


FIGURE 9. Mappings between context features and CUI model fragments.

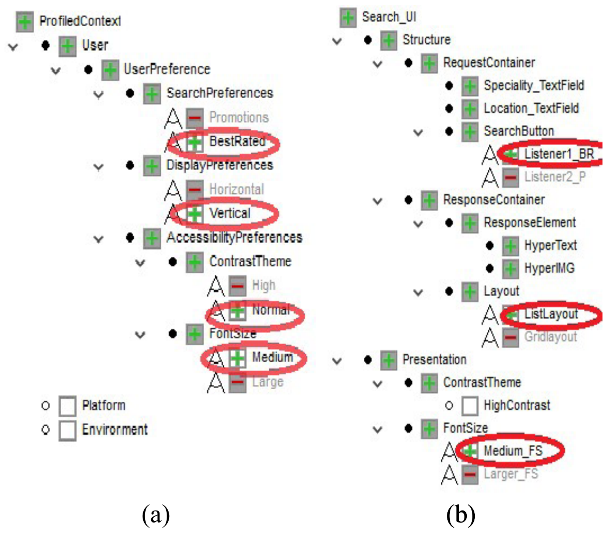


FIGURE 10. The feature model configuration. (a) The Context F.M configuration. (b) The UI feature model configuration.

TABLE 2. Feature constraints relative to the default context feature model of figure 4.

Event	Condition	Action
BestRated, vertical, normal, medium	Best Rated	Listener_BR
BestRated, vertical, normal, medium	Vertical	ListLayout
BestRated, vertical, normal, medium	Normal	¬HighContrast
BestRated, vertical, normal, medium	Medium	Medium_FS

composed by three objects, namely, the main container (the window1 object), the div container (the div1 object), and the label object (the searchpreference object). The search-preferences object is contained (which justifies the aggregation association) in the div1 container and this later in contained (another aggregation association), in turn, in the window1 container object.

Equally, the promotions context feature is mapped into a CUI model fragment. This later is composed by four objects: a window object (window1), a div container object (div1), a combobox object (CB1), and a label object (promotions). For the object relationship, the window1 object contains (an aggregation relationship) the div1 object which contains (an aggregation relationship), in turn, the CB1 object. This later contains (another aggregation relationship) the promotions object.

C. APPLICATION ANALYSIS

1) FEATURE MODEL CONFIGURATION

In order to generate initial interfaces (context and search UIs), first of all, we have to configure the application feature models. The context feature model was configured according to a default context of use. In our case, the default context is defined as follows (figure 10 (a)): the user prefers visualizing the **best rated** restaurants displayed **vertically** in a **normal** contrast theme and with a **medium** font size.

Secondly, to configure the search UI feature model, we use the default configuration of the context feature model and the appropriate feature constraints (i.e. feature constraints defined according to the selected context features, table 2). So according to the prescribed default context feature model and using the features constraints of table, the obtained configuration of the UI feature model

As depicted in figure 10(b), the default search UI feature model configuration is defined as follows: for the search button, we have selected the Listener_Br feature. This latter looks for best rated restaurants. For the response container layout, we have selected as default layout the listlayout. This latter allows displaying the result in a vertical way. And for presentation features, we selected as theme, the normal-Contrast contrast theme and as font size, the medium_FS feature.

D. APPLICATION IMPLEMENTATION

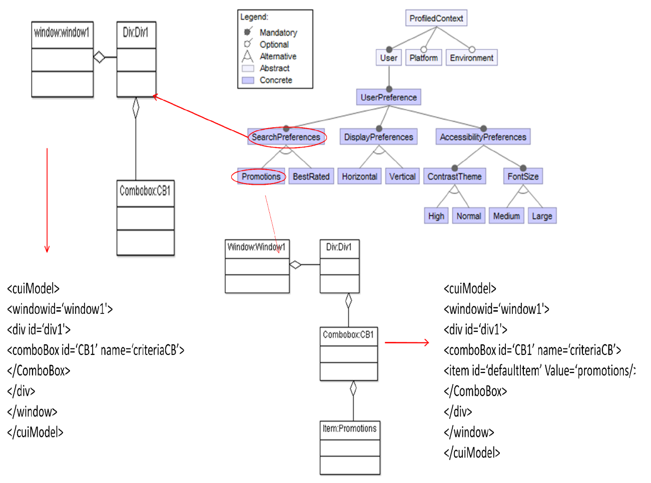
1) USER INTERFACE COMPOSITION

After feature model configuration, the next step is the composition of features artifacts. Feature artifacts are CUI models which correspond to the selected features. We have to compose the context features and the search UI features in order to generate consecutively the context UI and the search UI.

To perform feature artifacts composition, we took a look at existing model composers. We note that some existing model composer [2], [18] are still at the prototype stage. So to compose our CUI models, we opted for the conversion of these models into XML files (figure 11), then we used an XML file merger as model composer. As described in [37], the merge script is used to merge two XML files. For that, we use Algorithm1 (described below) with the aim to merge the XML files representing the selected feature of the context feature model and the search UI feature model.

Algorithm 1 Composition Algorithm

```
Require: A set of features artifacts {artifact}
Require: The current search UI configuration Pcurrent = {F1,F2, . . ., Fk}
Ensure: The merged XML file (RF.xml)
1: RF.xml = {the first artifact to be merged}
2: for all (f ∈ Pcurrent) do
3:     RF.xml = merge (artifact(F), RF.xml)
4: end for
```



```
<cuiModel>
<windowid='window1'>
<div id='div1'>
<comboBox id='CB1' name='criteriaCB'>
</ComboBox>
</div>
</window>
</cuiModel>
```

```
<cuiModel>
<windowid='window1'>
<div id='div1'>
<comboBox id='CB1' name='criteriaCB'>
<item id='defaultItem' Value='promotions/:
</ComboBox>
</div>
</window>
</cuiModel>
```

FIGURE 11. Feature mappings and the conversion of the CUI model into XML code.

Algorithm1 illustrates how a UI is recomposed using the artifacts which corresponds to the selected UI features. To start, the algorithm requires the current configuration, the appropriate implementation artifacts. The algorithm consists in browsing the list of selected features (saved in the new configuration file). For each feature, we merge its artifact with the already merged artifacts saved in the merge file (RF.xml). The algorithm iterates over features composing the new configuration file.

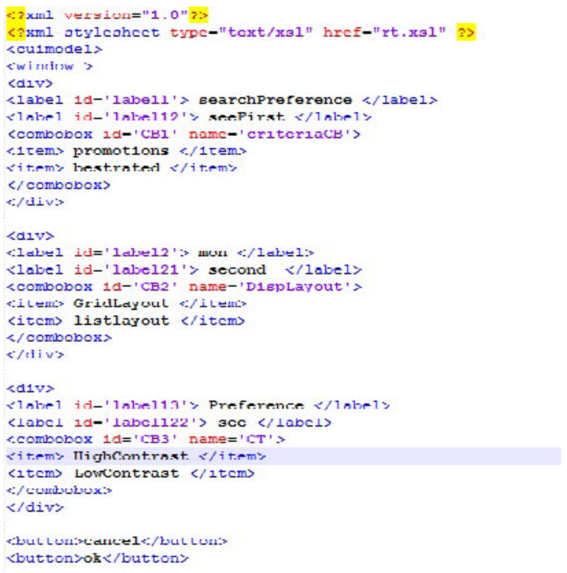
Using the prescribed algorithm, the result of composition of context features is the XML file described in figure 12.

2) FINAL USER INTERFACES

To generate final UIs, we used the Extensible StyleSheet Language Transformations (XSLT). The XSLT language transforms the XML file (resulted from CUI models composition) into a HTML page. Figure 13 (a) depicts the XSLT file which transforms the XML file related to the context page to a HTML page and figure 13(b) depicts the obtained HTML page presenting the context UI.

VI. THE RUNTIME PHASE IMPLEMENTATION

In this section, we give more details about the runtime adaptation mechanism. In figure 14, we highlight the architecture of the runtime adaptation mechanism. In this architecture, we specify, in particular, runtime components, used algorithms and data storages.



```
<?xml version="1.0"?>
<?xml stylesheet type="text/xsl" href="rt.xsl" ?>
<cuimodel>
<window >
<div>
<label id='label1'> searchPreference </label>
<label id='label12'> seeFirst </label>
<combobox id='CB1' name='criteriaCB'>
<item> promotions </item>
<item> bestrated </item>
</combobox>
</div>

<div>
<label id='label2'> mon </label>
<label id='label21'> second  </label>
<combobox id='CB2' name='DispLayout'>
<item> GridLayout </item>
<item> listlayout </item>
</combobox>
</div>

<div>
<label id='label13'> Preference </label>
<label id='label122'> see </label>
<combobox id='CB3' name='CT'>
<item> HighContrast </item>
<item> LowContrast </item>
</combobox>
</div>

<button>cancel</button>
<button>ok</button>
```

FIGURE 12. The XML representation of the context UI.

A. THE CONTEXT MANAGER

The context data are acquired from the context interface and are saved into a text file in the form of the couple <contextdata, Value> where the ‘‘contextdata’’ describes the context data description and the ‘‘value’’ is a Boolean value which is equal to true if the context data is selected by the user, else the context data value is equal to false. Figure 15 depicts the default context of use (figure 15(a)) and the new context of use (figure 15(b)).

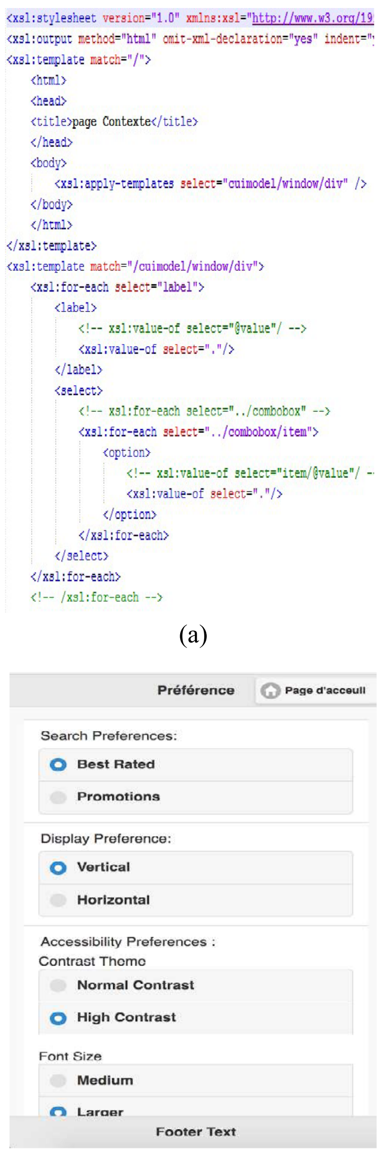
B. ADAPTATION MANAGER

The second runtime component is the adaptation manager component. This later encompasses two scripts. The first script is responsible for looking for a new configuration and the second script is responsible for the implementation of the new configuration and the recomposition of the new UI.

1) THE RECONFIGURATION ALGORITHM

To look for a new configuration, the reconfiguration algorithm requires as inputs, the set of context data, the running UI configuration (figure 16) and adaptation rules (figure 16).

Algorithm2 illustrates how the new feature configuration is obtained from context information. To start, the algorithm requires a set of updated observables, obtained from the context manager, the current configuration (also called the running configuration) obtained from the configuration storage and adaptation rules obtained from adaptation rules storage. The first step consists in creating a target configuration with the same variants as in the current configuration (line 1). Afterwards, the algorithm iterates over the updated observables. For the observables whose action belongs to the current configuration, the algorithm verifies if the new observable value is false. In that case, the action has to be unwoven from the target configuration. For the observables whose action does not belong to the configuration the algorithm verifies



```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/19
<xsl:output method="html" omit-xml-declaration="yes" indent="
<xsl:template match="/">
    <html>
    <head>
    <title>page Contexte</title>
    </head>
    <body>
        <xsl:apply-templates select="cuimodel/window/div" />
    </body>
    </html>
</xsl:template>
<xsl:template match="/cuimodel/window/div">
    <xsl:for-each select="label">
        <label>
            <!-- xsl:value-of select="@value"/ -->
            <xsl:value-of select="."/>
        </label>
        <select>
            <!-- xsl:for-each select="../combobox" -->
            <xsl:for-each select="../combobox/item">
                <option>
                    <!-- xsl:value-of select="item/@value"/ -
                    <xsl:value-of select="."/>
                </option>
            </xsl:for-each>
        </select>
    </xsl:for-each>
    <!-- /xsl:for-each -->
```

(a)

Préférence | Page d'acceuil

Search Preferences:
- Best Rated
- Promotions

Display Preference:
- Vertical
- Horizontal

Accessibility Preferences :
Contrast Theme
- Normal Contrast
- High Contrast

Font Size
- Medium
- Larger

Footer Text

(b)

FIGURE 13. **The generation of the context interface. (a) Excerpt of the XSLT transformation file. (b) The context HTML page.**

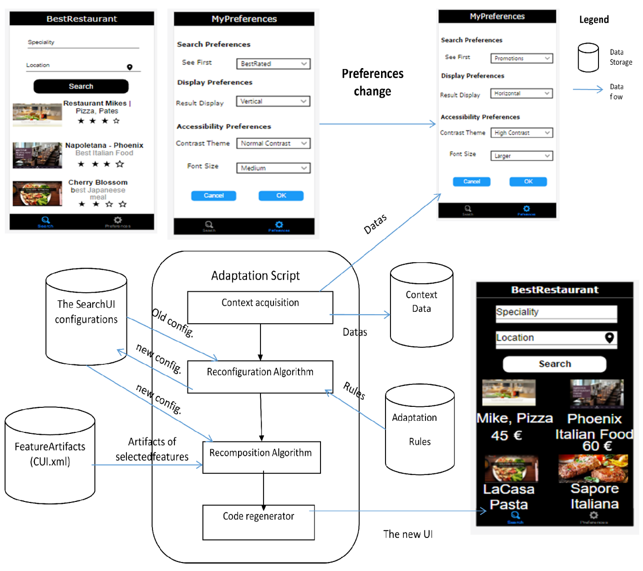
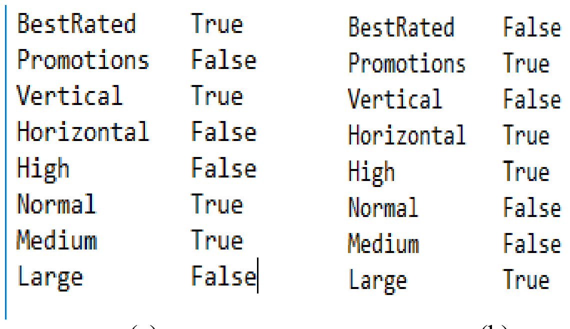


FIGURE 14. **The runtime adaptation mechanism.**



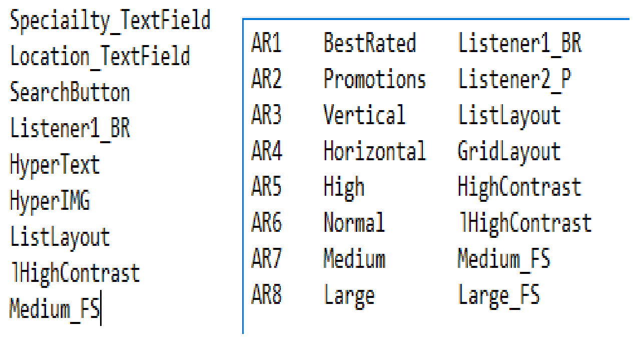
BestRated	True
Promotions	False
Vertical	True
Horizontal	False
High	False
Normal	True
Medium	True
Large	False

(a)

BestRated	False
Promotions	True
Vertical	False
Horizontal	True
High	True
Normal	False
Medium	False
Large	True

(b)

FIGURE 15. **Context data. (a) The default context of use. (b) The new context of use.**



```
Speciailty_TextField
Location_TextField
SearchButton
Listener1_BR
HyperText
HyperIMG
ListLayout
lHighContrast
Medium_FS
```

(a)

AR1	BestRated	Listener1_BR
AR2	Promotions	Listener2_P
AR3	Vertical	ListLayout
AR4	Horizontal	GridLayout
AR5	High	HighContrast
AR6	Normal	lHighContrast
AR7	Medium	Medium_FS
AR8	Large	Large_FS

(b)

FIGURE 16. **The Inputs of the reconfiguration algorithm. (a) The running configuration. (b) Adaptation Rules.**

if the new observable value is true. In that case, the action has to be woven to the target configuration.

2) THE INTERFACE REGENERATION

After feature reconfiguration, we recompose the selected features using Algorithm 1 as described in the design phase. The result of the runtime recomposition is an XML file which describes the new search UI.

Like at the design phase, the code generation is performed using the XSLT transformation language. The XSLT file is applied on the XML file resulted from the composition of the new UI. As shown in figure 18, the first interface (figure 17 (a)) results from the design phase and represents the search UI before its adaptation. The second interface (figure 17(b)) is the first adaptation of the search UI. It corresponds to the context 1 described as follows: the user prefers visualizing the restaurants in promotions, displayed in an horizontal way with a contrast theme and a large font size.

The third interface (figure 17(c)) is another adaptation for the search UI which corresponds to the context 2 described as

Algorithm 2 Runtime Reconfiguration

```
Adapter algorithm Require: A set of updated context observables C
Require: The current product configuration Pcurrent = {F1,F2, . . .,Fk}
Require: The set of adaptation rules AR
Ensure: A target product configuration Ptarget
1: Ptarget ← Pcurrent
2: for all (O ∈ C) do
3: if (AR(O) ∈ Pcurrent) then
4: if (O.value() = false) then
5: Ptarget.deselect(AR(O))
6: else
7: Ptarget.select (AR(O))
8: endif
9: end if
10: end for
```

follows: the user prefers visualizing the best rated restaurants, displayed in a vertical way with a contrast theme and a large font size.

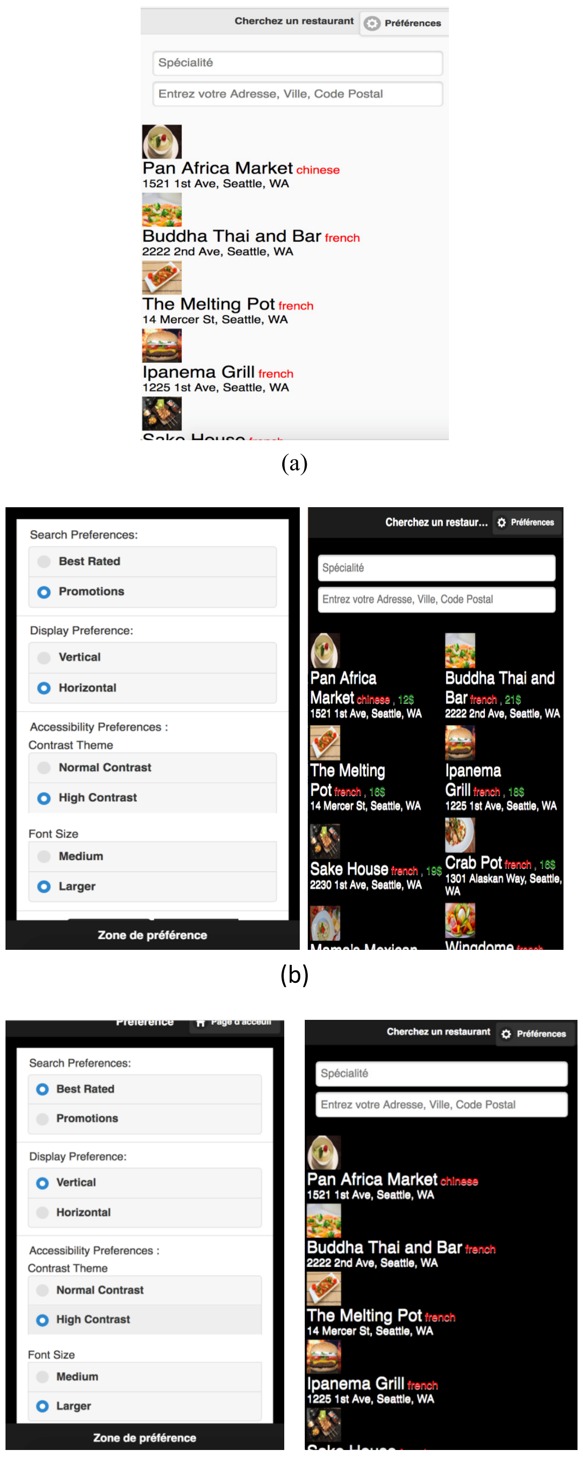
VII. EVALUATION AND DISCUSSION

To evaluate our approach, we proceed for a 1) scenario-based evaluation, a 2) scalability evaluation and for a 3) user evaluation of the generated interfaces. The first type of evaluation is based on the used illustrative case study. The second evaluation is based on two metrics (the generation time, the adaptation time) and user's evaluation is based on a questionnaire to which users must respond. For a better evaluation, this section includes a comparison of our approach with existing approaches [12], [19], [20].

For the scenario based evaluation and as described in section 2, the illustrative SPL is relative to a "search for restaurant" application. To manage the application variability, we have defined two variability models: the context feature model and the UI feature model. The variability of the UI feature model was defined across different UI aspects such as: UI elements (speciality textField, location textField and the hyperlink response objects), presentation units (request container, response container), the visual aspect (text font size, UI contrast theme) and the layout aspect (response hyperlinks displayed in the form of grid, response hyperlinks displayed vertically).

The context variability was defined across three UI aspects: UI element (display the best rated restaurants or the restaurants in promotions), the layout (response hyperlinks displayed using a gridlayout, response hyperlinks displayed using a listlayout) and the visual appearance of the UI (text size, UI contrast theme).

To implement context and UI variabilities, we used the CUI models, each concrete feature is associated to a fragment of the CUI diagram (instantiated from the meta-model of figure 6). Each artifact is, then converted into their XML representation then composed with the others artifacts in order to generate the search UI and the context UI.


(a)

Cherchez un restaurant | Préférences

Spécialité

Entrez votre Adresse, Ville, Code Postal

Pan Africa Market chinese
1521 1st Ave, Seattle, WA

Buddha Thai and Bar french
2222 2nd Ave, Seattle, WA

The Melting Pot french
14 Mercer St, Seattle, WA

Ipanema Grill french
1225 1st Ave, Seattle, WA

(b)

Search Preferences: Best Rated [✓ Promotions]

Display Preference: Vertical [✓ Horizontal]

Accessibility Preferences : Contrast Theme: Normal Contrast [✓ High Contrast]

Font Size: Medium [✓ Larger]

Zone de préférence

Cherchez un restaur... | Préférences

Spécialité

Entrez votre Adresse, Ville, Code Postal

Pan Africa Market chinese , 12$
1521 1st Ave, Seattle, WA

Buddha Thai and Bar french , 21$
2222 2nd Ave, Seattle, WA

The Melting Pot french , 18$
14 Mercer St, Seattle, WA

Ipanema Grill french , 18$
1225 1st Ave, Seattle, WA

Sake House french , 19$
2230 1st Ave, Seattle, WA

Crab Pot french , 18$
1301 Alaskan Way, Seattle, WA

(c)

Preference | Page d'accueil

Search Preferences: [✓ Best Rated] Promotions

Display Preference: [✓ Vertical] Horizontal

Accessibility Preferences : Contrast Theme: Normal Contrast [✓ High Contrast]

Font Size: Medium [✓ Larger]

Zone de préférence

Cherchez un restaurant | Préférences

Spécialité

Entrez votre Adresse, Ville, Code Postal

Pan Africa Market chinese
1521 1st Ave, Seattle, WA

Buddha Thai and Bar french
2222 2nd Ave, Seattle, WA

The Melting Pot french
14 Mercer St, Seattle, WA

Ipanema Grill french
1225 1st Ave, Seattle, WA

FIGURE 17. The adaptation of the search UI. (a) The search UI before adaptation. (b) Search UI adaptation – context1. (c) Search UI adaptation-Context 2.

Regarding kramer approach, the illustrative SPL is about a content store case study. In this case study, the store can distribute different content such as video or music. The content can be distributed depending on the location of the device. The UI variability was defined according to GUI elements, GUI elements properties and GUI behavior. To implement the UI variability, the author opted for the document-oriented technology. After the composition of document artifacts, the resulted document is interpreted on an Android platform.

In Gabillon's approach, the use case was about a dashboard user interface. The UI variability was defined according to two aspects: presentation units and UI elements. To implement the UI variability, the author used component technology.

By comparing our approach with other approaches, we note that our approach took into consideration many aspects when describing the UI variability. Except for the presentation units and UI elements, we have described the UI variability according to the layout and the visual appearance (text size, UI color) aspects. The design and the implementation of such features (layout feature, visual appearance feature) ensure the generation of a more ergonomic interface and make the UI development easier and the modification of some properties simpler. Furthermore, and contrary to Kramer and Gabillon approaches which respectively use document-based technology and component-based technology, the use of models as implementation technology makes the design phase process more reusable and more abstract.

At the runtime phase, by comparing our approach with the other approaches, we note that the principal difference is about the context acquisition. In our approach, the context was entered manually. The main UI is adapted according to the context data updated by the end-user. The first context was about a user who prefers visualizing the restaurants in promotions, displayed in a horizontal way with a contrast theme and large font size. And the second context is about a user who prefers visualizing the best rated restaurants, displayed in a vertical way and with a contrast theme and a large font size. In Kramer approach, the adaptation was about some platform properties such as the actual position, connectivity and the battery while in Gabillon's approach, the context was about the screen size.

For the scalability evaluation, we opted for the use of the generation time and the adaptation time. In table 3, we present the two metrics and the correspondent values for our approach and Kramer's approach. Gabillon did not carry out a scalability evaluation.

- **Generation Time**: this is the time the tool takes to generate the source code of initial interfaces. For this metric, we do not consider the time required for feature model design or for artifacts implementation. We only consider the time of feature model configuration, UIs composition and the generation of the source code. In our approach, the generation time was calculated for the context UI and the search UI. As depicted in table 3, the generation time of the context UI varies between 0.86s (for 1 feature) to 13.05 mins (for 10 features) while for the search UI, the generation time varies between 0.89s (for 1 feature) to 14.57 mins (for 14 features). For Kramer's approach, the generation time varies between 1.11s (for 1 feature) to 23.43 mins (for 14 features).
- **Adaptation Time**: this metric measure the time it takes for a complete adaptation cycle to take place. This time assumes a new configuration, the recomposition of the new UI and the time for the generation of the HTML source code. In our approach, to measure the adaptation time, we adapt the application 10 times to fetch the average time for these adaptations. The average time is equal to 2.03 ms (for 1 feature) and to 68.33 ms (for 14 features). In Kramer's approach, the adaptation time is equal to 2.44ms (for 1 feature) to 63.72ms (for 14 features). At the runtime phase, Kramer's approach is more rapid. The reason is that the recomposition (at the runtime) is performed by weaving/unweaving document and not by recomposing all feature artifacts as is the case of our approach, which shows the limit of our approach.

TABLE 3. Scalability evaluation results.

Approach	Metrics	
	Generation Time	**Adaptation Time**
Kramer's Approach	Min. Time=1.11s Max. Time =23.43m	Min. Time=2.44ms Max. Time=63.72ms

Sboui's Approach	**Context interface**
	Min.Time=0.86s/Max.Time=13.05m
	Main interface
	Min.Time=0.89s/Max.Time=14.57m
		Min. Time =2.03ms
		Max. Time =68.33ms

Gabillon's Approach	_	_

The last evaluation is a qualitative evaluation, for that we sent an on-line questionnaire to 11 participants (5 females and 6 males). These participants were coming from 3 different countries belonging to 2 different continents with different types of background and occupation (e.g., students in computer science, teachers and students in various languages, as well as researchers in social and exact sciences). The participants were aged between 22 and 43. The evaluation includes a debriefing questionnaire based on the IBM Computer Satisfaction Usability Questionnaire (CSUQ) [21].

The CSUQ questionnaire is an empirically-validated 19-question questionnaire benefiting from an $\alpha = 0.89$ reliability coefficient related to usability, thus meaning that answers provided by participants to this questionnaire demonstrate a high correlation with the usability of the system being evaluated. Each IBM CSUQ closed question was measured using a 7-point Likert scale (1 = strongly disagree, 2 = largely disagree, 3 = di-sagree, 4 = neutral, 5 = agree, 6 = largely agree, 7 = strongly agree) and was phrased positively as follows:

1. Q1 : Overall, I am satisfied with how easy it is to use this model.

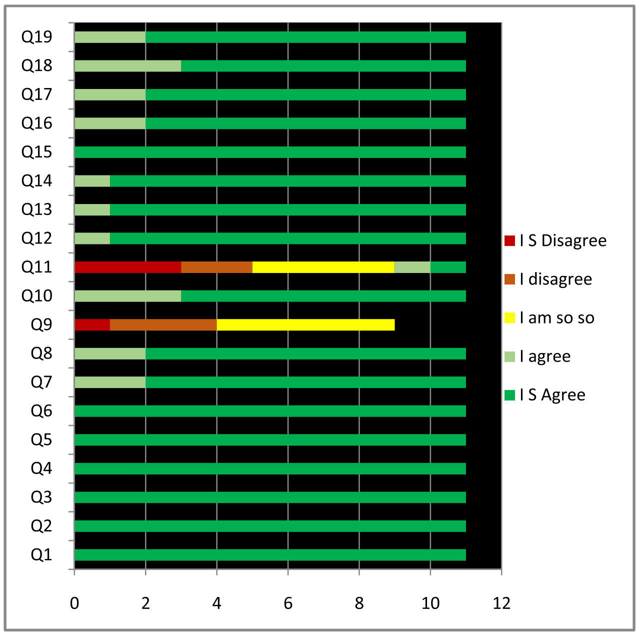


FIGURE 18. Distribution of participants' answers to the IBM CSUQ questionnaire.

2. Q2 : It was simple to apply this model.
3. Q3 : I can effectively complete my task applying this model.
4. Q4 : I am able to complete my task quickly applying this model.

Q5 : I am able to efficiently complete my task applying this model.

6. Q6 : I feel comfortable applying this model.
7. Q7 : It was easy to learn to applying this model.
8. Q8 : I believe I became productive quickly applying this model.
9. Q9 : The model provides me with structured guidance on how to fix problems.
10. Q10 : Whenever I make a mistake using the model, I recover easily and quickly.
11. Q11 : The information provided by the model and its accompanying method is clear.
12. Q12 : It is easy to find the information I needed.
13. Q13 : The information provided for the model is easy to understand.
14. Q14 : The information is effective in helping me complete the tasks and scenarios.
15. Q15 : The organization of information on the model screens is clear.
16. Q16 : The interface of this model is pleasant.
17. Q17 : I like using the interface of this model.
18. Q18 : This model has all the functions and capabilities I expect it to have.
19. Q19 : I am satisfied in using this model.

Questionnaire Results: Figure 18 graphically depicts the distribution of the answers provided by the participants on the 19 IBM CSUQ questions. Each cumulated horizontal histogram of Figure 18 could be interpreted as follows: a score between 6 and 7 represented with dark green, is considered as excellent; a score of 5, represented with light green, is considered as good; a score of 4, represented with yellow, is considered as average, a score of 3, represented in orange, is considered as poor; and a score between 1 and 2, represented in red, is considered very bad. In general, a score between 'average' and 'excellent' should not raise any particular concern regarding this question, whereas a score between 'poor' and 'very bad' should raise some discussion in order to investigate why this question has been depreciated so much. Figures 19 and 20 summarise the aggregated CSUQ sub-metrics reported in table 4. Each CSUQ questionnaire involves the calculation of four quality metrics of the system being evaluated as follows:

1. System usefulness (SysUse : Items 1-8)
2. Quality of the information (InfoQual : Items 9-15)
3. Quality of the interaction (InterQual : Items 16-18)
4. Overall quality of the system (Overall : Item 19)

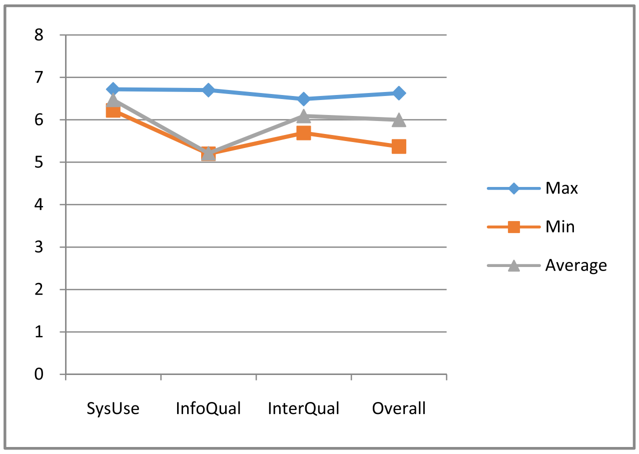


FIGURE 19. Aggregated scores by CSUQ sub-metrics (Min, Max, Average).

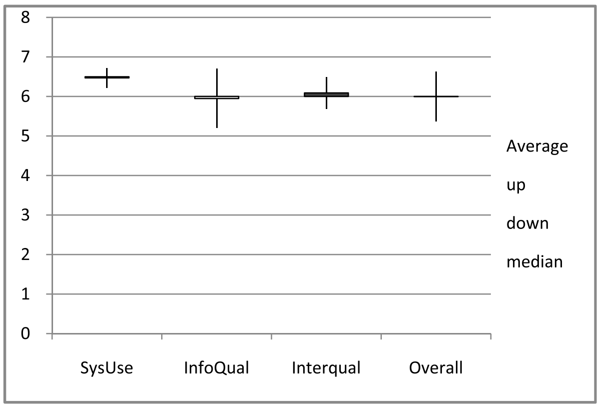


FIGURE 20. Aggregated scores by CSUQ sub-metrics(Average, Up, Down, Median).

Figure 18 suggests that the global subjective satisfaction of participants involved in the experiment follows a positive trend since Q19 is interpreted positively by 11 users out of 11 (Q19, $\mu = 6$, M = 6, s = 0.63). The most posi-

TABLE 4. Scores by CSUQ sub-metrics.

Sub-metric	Average	Mediane	Avg. Deviation	Std. Deviation
SysUse	6,47	6.5	0.84	6,5
InfoQual	4.79	6	1.05	0.8
Interqual	6,09	6	5	0.4
Overall	6	6,6	0.36	0.63

tively evaluated sub-metric is certainly the system usefulness (Q1- Q8, $\mu = 6.47$, M = 6.5, s = 0.25) : all eight questions do not have any negative answers, the average is the highest and the standard deviation is the smallest, thus suggesting that respondents tend to agree that the whole system is very useful to them. Second comes the interaction quality (Q16-Q18, $\mu = 6.09$, M = 6, s = 0.4): the average is considered high as well as the median with small deviation. Next comes the information quality (Q9-Q15, $\mu = 4.79$, M = 6, s = 0.8) : some questions have negative answers, the average is lower with a more disperse variance, thus indicating that there is no strong agreement among the respondents regarding to this sub-metric. Questions Q9 and Q11 raise a particular concern. They are the only questions receiving strong disagreement. Q9, "The model provides me with structured guidance on how to fix problems" indicates that the system does not guide enough the users to correct an error when this latter is produced. Q11, "The information provided by the model and its accompanying method is clear" indicates that the help messages are not clear and do not help the user sufficiently.

The analysis of the CSUQ sub-metrics in Table 4 and Figures 19 and 20 evidences that participants perceived as "useful" the application ($\mu = 6.47$, M = 6.5, s = 0.25) and said to be "Overall satisfied" ($\mu = 6$, M = 6, s = 0.63).

VIII. CONCLUSION

Our work is considered as a new contribution in the area of adaptation of user interfaces to the context of use. In this paper, we have proposed a UI-DSPL approach for the development of context-adaptable UIs. The main contributions of our approach are 1) target the user preference to adapt the main interface. 2) Combine MBUID and DSPL concepts to make the UI-DSPL approach more abstract and more reusable 3) propose a design pattern to facilitate the design and the development of the runtime adaptation mechanism.

In the paper, we have highlighted the approach phases and their implementations. The design phase is dedicated for the generation of initial UIs and a runtime phase is dedicated for the adaptation of the main UI according to the user preference.

To validate the proposed approach, we have used two types of evaluation. The first evaluation is a scalability evaluation in which we have evaluated the generation time and the adaptation time of the restaurant search application. The second evaluation is a qualitative/quantitative evaluation based on the IBM CSUQ questionnaire.

For future works, we will use the machine learning technique. This latter puts the user at ease by choosing the interface that suits him. Furthermore, we will promote the development process according to the results of the questionnaire, we will evaluate the approach by designers using the NASA Task Load Index (TLX) questionnaire [15] and we will consider more context elements while adapting our interfaces (e.g. the platform).

REFERENCES

[1] S. Apel and C. Kästner, "An overview of feature-oriented software development," *J. Object Technol.*, vol. 8, no. 5, pp. 49–84, 2009.

[2] S. Apel, C. Kästner, and J. Liebig. *FeatureHouse: Language-Independent, Automated Software Composition*. Accessed: Sep. 2, 2016. [Online]. Available: http://www.infosun.fim.uni-passau.de/spl/apel/fh/

[3] H. Arboleda, A. Romero, R. Casallas, and J.-C. Royer, "Product derivation in a model-driven software product line using decision models," in *Proc. 12th Conf. Iberoamericana Soft. Eng. (CIbSE)*, Medellín, Colombia, Apr. 2009, pp. 59–72. [Online]. Available: http://ai2-s2-pdfs.s3.amazonaws.com/57ae/634647fcf7e610df3d106eb2a3cd0f152733.pdf

[4] W. Bouchelligua, A. Mahfouthi, and L. Benammar, "An MDE approach for user interface adaptation to the context of use," in *Proc. 3rd Int. Conf. HCSE*, Reykjavik, Iceland, Oct. 2010, pp. 62–78.

[5] Q. Boucher, G. Perrouin, and P. Heymans, "Deriving configuration interfaces from feature models: A vision paper," in *Proc. 6th Int. Workshop Variability Model. Softw.-Intensive Syst.*, Jan. 2012, pp. 37–44.

[6] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, "A unifying reference framework for multi-target user interfaces," *Interact. Comput.*, vol. 15, no. 3, pp. 289–308, Jun. 2003.

[7] R. Capilla, J. Bosch, P. Trinidad, A. Ruiz-Cortés, and M. Hinchey, "An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry," *J. Syst. Softw.*, vol. 91, pp. 3–23, May 2014.

[8] T. Cerny, K. Cemus, M. J. Donahoo, and E. Song, "Aspect-driven, data-reflective and context-aware user interface," *ACM SIGAPP Appl. Comput. Rev.*, vol. 13, no. 4, pp. 53–66, Dec. 2013.

[9] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Boston, MA, USA: Addison-Wesley, 2002.

[10] *Eclipse Modeling Framework*. Accessed: Oct. 10, 2016. [Online]. Available: http://www.eclipse.org/modeling/emf

[11] J. D. A. S. Eleutério and C. M. F. Rubira, "A comparative study of dynamic software product line solutions for building self-adaptive systems," Comput. Inst., Univ. Campinas, Portugal, Tech. Rep. C-17-05, 2017.

[12] Y. Gabillon, N. Biri, and B. Otjacques, "Designing an adaptive user interface according to software product line engineering," in *Proc. ACHI*, vol. 15. 2015, pp. 86–91.

[13] K. Z. Gajos, D. S. Weld, and J. O. Wobbrock, "Automatically generating personalized user interfaces with supple," *Artif. Intell.*, vol. 174, nos. 12–13, pp. 910–950, 2010.

[14] K. Garcés, C. Parra, H. Arboleda, A. Yie, and R. Casallas, "Variability management in a model-driven software product line," *Revista Avances Sistemas Informática*, vol. 4, no. 2, pp. 3–12, Sep. 2007.

[15] V. J. Gawron. (2000). *Human Performance Measures Handbook, Nasa TLX Questionnaire*. [Online]. Available: http://theses.univ-lyon2.fr/documents/getpart.php?id=lyon2.2010.maincent_a&part=365749

[16] H. Gomaa and M. Hussein, "Dynamic software reconfiguration in software product families," in *Proc. PFE*, 2003, pp. 435–444.

[17] J. M. G. Calleros *et al.* (May 2010). *Model-Based UI XG Final Report*. [Online]. Available: https://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/

[18] F. Heidenreich. (Sep. 2013). *FeatureMapper Mapping Features to Models*. [Online]. Available: http://featuremapper.org

[19] D. M. Kramer, "Unified gui adaptation in dynamic software product lines," Ph.D. dissertation, Dept. Comput., Univ. West London, London, U.K., 2014.

[20] D. Kramer, S. Oussena, P. Komisarczuk, and T. Clark, "Using document-oriented GUIs in dynamic software product lines," *ACM SIGPLAN Notices*, vol. 49, no. 3, pp. 85–94, Mar. 2014. [Online]. Available: https://doi.org/10.1145/2637365.2517214
[21] J. R. Lewis. *Questionnaire sur l'usabilité d'un Système Informatisé*. Accessed: Sep. 21, 2017. [Online]. Available: http://garyperlman.com/quest/quest.cgi
[22] I. M. S. Logre, P. Collet, and M. Riveilli, "Sensor data visualisation: A composition-based approach to support domain variability," in *Proc. 10th Eur. Conf. Model. Found. Appl. (ECMFA*, vol. 8569. New York, NY, USA, Jul. 2014, pp. 101–116. [Online]. Available: https://doi.org/10.1007/978-3-319-09195-2_7
[23] N. Mezhoudi, "User interface adaptation based on user feedback and machine learning," in *Proc. Companion Pub. Int. Conf. Intell. Interfaces Companion (IUI)*, New York, NY, USA, 2013, pp. 25–28, doi: 10.1145/2451176.2451184.
[24] J. Müller, "Generating graphical user interfaces for software product lines: A constraint-based approach," in *Forschungsberichte des Instituts für Wirtschaftsinformatik der Universität Leipzig/15. Interuniversitäres Doktorandenseminar Wirtschaftsinformatik der Universitäten Chemnitz, Dresden, Freiberg, Halle-Wittenberg, Jena und Leipzig*. Qucosa, Germany: Electronic Server, 2011, pp. 64–71. [Online]. Available: https://pdfs.semanticscholar.org/186d/7f0907852cbaaf798513ea1f2e347a63b342.pdf
[25] A. Pleuss, B. Hauptmann, M. Keunecke, and G. Botterweck, "A case study on variability in user interfaces," in *Proc. 16th Int. Softw. Product Line Conf.*, vol. 1. New York, NY, USA, 2012, pp. 6–10.
[26] A. Pleuss, S. Wollny, and G. Botterweck, "Model-driven development and evolution of customized user interfaces," in *Proc. 5th ACM SIGCHI Symp. Eng. Interact. Comput. Syst.*, Jun. 2013, pp. 13–22.
[27] A. Pleuss, B. Hauptmann, D. Dhungana, and G. Botterweck, "User interface engineering for software product lines: The dilemma between automation and usability," in *Proc. 4th ACM SIGCHI Symp. Eng. Interact. Comput. Syst. (EICS)*, Copenhagen, Denmark, Jun. 2012, pp. 25–34. [Online]. Available: https://doi.org/10.1145/2305484.2305491
[28] K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. New York, NY, USA: Springer, 2005.
[29] C. Quinton, S. Mosser, C. Parra, and L. Duchien, "Using multiple feature models to design applications for mobile phones," in *Proc. SPLC*, Munich, Germany, Aug. 2011, pp. 1–8.
[30] M. Schlee and J. Vanderdonckt, "Generative programming of graphical user interfaces," in *Proc. 7th Int. Working Conf. Adv. Vis. Interfaces (AVI)*, Gallipoli, Italy, May 2004, 2004, pp. 403–406. [Online]. Available: https://doi.org/10.1145/989863.98993
[31] *Software & Systems Process Engineering Metamodel Specification*. Accessed: Sep. 22, 2017. [Online]. Available: http://www.omg.org/spec/SPEM/2.0/
[32] T. Sboui, A. Ben Ayed, and M. A. Alimi, "A meta-model for run time adaptation in a UI-DSPL process," in *Proc. BHCI*, 2017, pp. 1–7.
[33] T. Sboui, "A DSPL approach for the development of context-adaptable user interfaces," in *Proc. RCIS*, 2017, pp. 421–426.
[34] J. S. Sottet, A. Vagner, and A. G. Frey, "Model transformation configuration and variability management for user interface design," in *Proc. 3rd Int. Conf. Model-Driven Eng. Softw. Develop. (MODELSWARD)*, vol. 580. Angers, France, Feb. 2015, pp. 390–404. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-27869-8_23
[35] J.-S. Sottet *et al.*, "Model-driven adaptation for plastic user interfaces," in *Human-Computer Interaction*. Berlin, Germany: Springer, 2007, pp. 397–410.
[36] *UsiXML: User Interface eXtensible Markup Language*. Accessed: Oct. 20, 2016. [Online]. Available: http://www.w3.org/2005/Incubator/model-basedui/wiki/UsiXML
[37] (2002). *XSLT Stylesheets Useful Things and Other Jokes*. [Online]. Available: http://web.archive.org/web/20160809092524/http://www2.informatik.hu-berlin.de/~obecker/XSLT/

THOURAYA SBOUI is currently pursuing the Ph.D. degree. She is currently a Computer Science Teacher. She is member of the research group in Intelligent Machines Laboratory. She is currently an investigating User Interface Adaptation and Software Product Line Engineering.

MOUNIR BEN AYED received the Habilitation degree in computer system engineering from the Engineering School of Sfax, Tunisia, in 2013, the Ph.D. degree in biomedical engineering from Paris XII University, France, and the Postgraduate Diploma degree in biomedical engineering from the University of Technology of Compiègne, France.

He is currently an Associate Professor with the Computer Science Department, Faculty of Science, University of Sfax. His research activities concern decision support system based on a knowledge discovery from data process. Most of his research are designed and evaluated in the medical field. He received the University Accreditation for his habilitation research in computer system engineering from the Engineering School of Sfax.

ADEL M. ALIMI received the Degree in electrical engineering in 1990 and the Ph.D. and HDR degrees in electrical and computer engineering in 1995 and 2000, respectively.

He is currently a Professor in electrical and computer engineering with the University of Sfax.

His research interests include applications of intelligent methods (neural networks, fuzzy logic, and evolutionary algorithms) to pattern recognition, robotic systems, vision systems, and industrial processes. His research interest include intelligent pattern recognition, learning, analysis and intelligent control of large scale complex systems.