A Meta-model for Run Time Adaptation in a UI-DSPL process

Thouraya SBOUI
REGIM Lab.
National Engineering School
University of Sfax, Sfax, Tunisia
Sboui.thouraya@gmail.Com

Mounir BEN AYED
REGIM Lab.
Faculty of Sciences
University of Sfax, Sfax, Tunisia
Mounir.benayed@ieee.org

Adel M. ALIMI
REGIM Lab.
National Engineering School
University of Sfax, Sfax, Tunisia
adel.alimi@ieee.org

Unlike the conventional Software Product Line (SPL) process, the Dynamic Software Product Line (DSPL) process continues to reconfigure and adapt at runtime. In the context of the development of a family of adaptable user interfaces (UIs) and in order to facilitate the design and the development of the runtime adaptation mechanism, we propose a model which defines fundamental concepts required by the UI adaptation mechanism. The model aims to support user interface designers to develop and conceptualize system that accommodate context-awareness, and dynamic runtime adaptation requirements.

Adaptation Modeling; Run-Time UI Adaptation; UI-DSPL approach.

1. INTRODUCTION

From one hand, effective adaptation of User Interfaces (UI) is still a main requirement to improve system usability. The heterogeneity in contexts of use augmented the complexity of such a task. Design time adaptations are no more sufficient to guarantee context awareness. The support of context of use change is always needed at the run-time phase.

From another hand and to develop a family of user interfaces, recent works had resorted to the use of Software Product Line Engineering (SPLE) paradigm (Pleuss et al. 2013). The SPLE (Pohl et al., 2005) 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, in the context of development of adaptable UIs, some works opted for the use of Dynamic Software Product Line engineering (DSPLE) (Gabillon et al. 2015) (Kramer 2014). The DSPLE (Gomaa et Hussein, 2003) (Capilla et al., 2014) exploits the knowledge acquired in Software Product Line Engineering to develop systems that can be context-aware, post-deployment reconfigurable, or runtime adaptive.

The purpose of this research is to support runtime adaptation while considering the context of use change at runtime. We achieve this propose by proposing a state transition model that enhances context awareness and runtime adaptation. The paper is structured as follow: Section 2 presents an overview of UI adaptation using a software product line process. Section3 presents the proposed meta-model. Section 4 presents an illustrative use case. And section5 presents the meta-model instantiation.

2. STATE OF THE ART

To develop a family of UIs, many proposals opted for the use of Software Product Line Engineering (SPLE) paradigm (Pohl et al., 2005). This section presents an overview of UI-SPL proposals. Proposals are compared in table 1 according to the following criteria:

- The approach type: specifies the type of the proposed approach, a SPL approach, a model driven SPL approach, a model based SPL approach, or a dynamic SPL approach;
- Context type: specifies the type of runtime acquisition (derived, sensed or profiled) of the context of use;
- Context element: specifies the context element which was targeted at the runtime phase. As reference, we use the standard triplet which defines the context of use (<user, platform, environment>);
- Adaptation technique: specifies the technique used to adapt the UI at the runtime;
- Adaptation model: does the approach propose an adaptation model?.

© The Authors. Published by BCS Learning and Development Ltd.

Approach	Approach type	Context type	Context Element	Runtime Adaptation Technique	Adaptation modeling
[Garcés et al. 2007]	*MD-SPL*	*None*	*None*	*None*	*None*
[Quinton et al. 2011]	*MD-SPL*	*None*	*None*	*None*	*None*
[Boucher et al. 2012]	*SPL*	*None*	*None*	*None*	*None*
[Pleuss et al. 2013]	*SPL*	*None*	*<environment, platform, user, **customer**>*	*None*	*None*
[Kramer D. M. 2014]	*DSPL*	*Sensed*	*<environment, **platform**, user>*	*Document-based compositionnal technique*	*None*
[Gabillon et al. 2015]	*DSPL*	*Sensed*	*<user, **platform**, environment>*	*Component-based compositionnal technique*	*None*

Table1: Overview of UI adaptaion in SPL process

In (Garcés et al., 2007), the authors propose a multi-level MD-SPL approach in which they weave SPL and Model Driven Engineering (MDE) (Schmidt, 2006) artifacts to generate a graphical user interface (GUI). In Garcés's approach, there is neither context consideration, nor GUI adaptation.

In (Quinton et al. 2011), the authors propose an automatic MD-SPL approach that generates UIs for mobile devices by merging feature assets. To bridge the gap between the application feature diagram and the device feature diagram, 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. In Quinton's approach, there is neither context consideration, nor GUI adaptation.

In (Boucher et al., 2012), authors use a Model Based User Interface Development (MBUID) (Gonzales-Carelos, 2010) models to implement UI features. Boucher process was reserved to develop configuration interfaces. In Boucher's approach, there is neither context consideration nor UI adaptation.

In (Pleuss et al., 2013), the context awareness was performed at the design time phase. In (Pleuss et al., 2013), authors used MBUID models to implement the SPL process. The three main elements that define the context of use (<user, platform, and environment>) were extended with the customer element. The customer is the person who buys the product. To support the customization of different UI aspects (Pleuss et al., 2012), the authors use manual models.

In (Kramer, 2014) and (Gabillon et al., 2015), authors propose a mixed adaptation. At the design time, the UI is developed using the current context of use and at the runtime, the UI is adapted according to the target context. For adaptation, it was not modelled. The authors of both approaches merely indicate the used adaptation technique. In (Kramer, 2014), the author uses a document-based compositional techniques while in (Gabillon et al., 2015), authors used a component-based compositional technique. For the context element, both proposals targeted the platform element.

Based on the above analysis, we note that:

- The context consideration within UI-SPL approaches was supported only by two proposals;
- To recompose the UI at the runtime, kramer's and Gabillon's approaches use different technologies (Kramer uses documents while Gabillon uses components);
- There is no proposition for a runtime adaptation model.

For that and in order to facilitate the design and the development of the runtime adaptation mechanism, we propose a specific model that describes the runtime adaptation within a UI-DSPL process. The model aims to unify the used runtime technique/technology and will serve as a design pattern for UI designers.

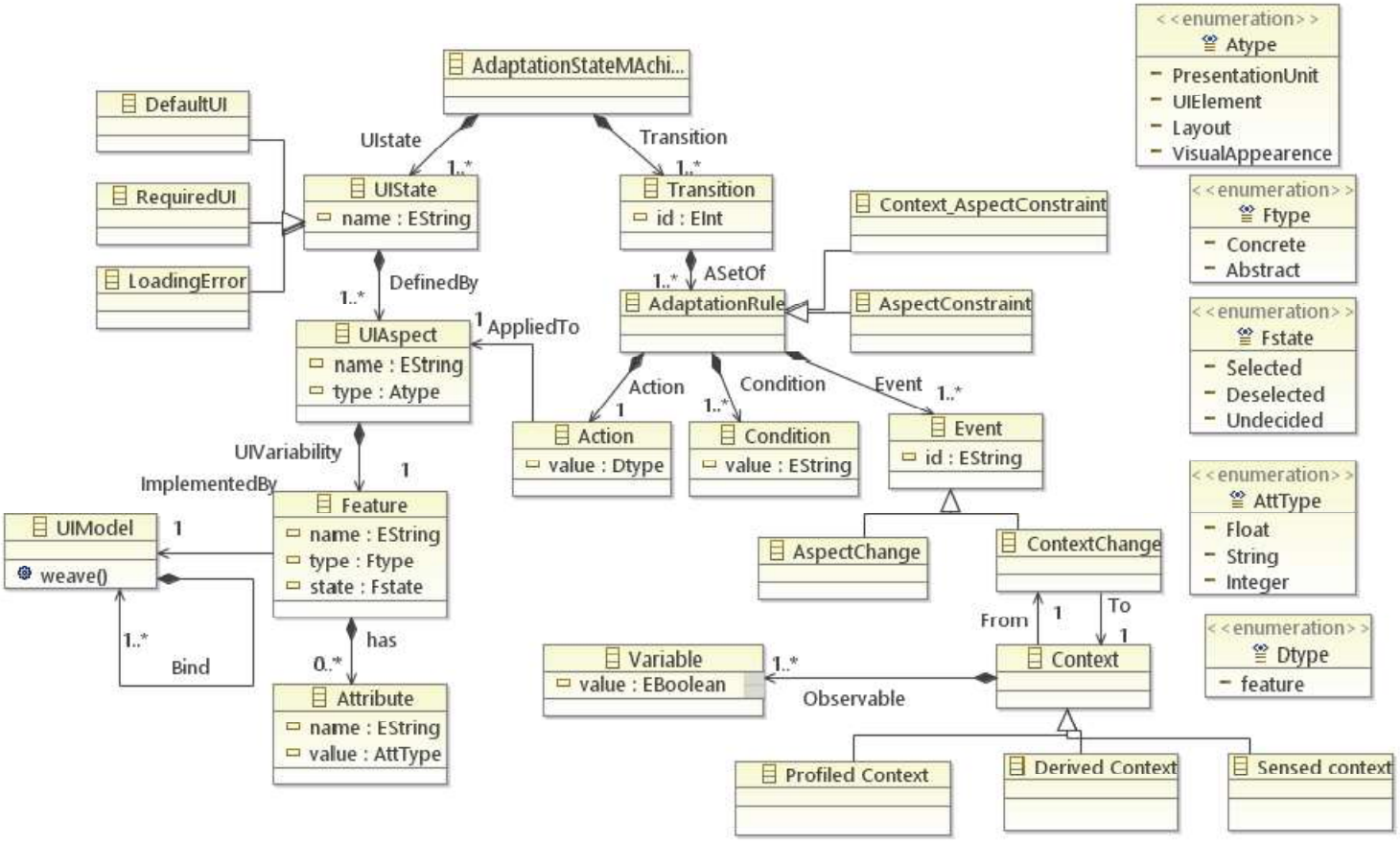


Figure1 : the EMF adaptation Meta-model

3. ADAPTATION METAMODEL

Before presenting our model, it is important to note that the runtime phase is preceded by a design phase. This phase is characterized by the definition of the following concepts:

- The UI Feature model: specify the variabilities and the commonalities of the user interface;
- Core asset: specify the technology used to implement features. To make our UI-SPL process more abstract and more reusable, we use as artifact the Model Based User Interface Development models;
- Feature constraints: describe the links between UI features and context features. Feature constraints will serve as an adaptation rules at the runtime phase.

Figure 1 depicts graphically the proposed model. The model highlights the fundamental concepts describing UI adaptation at the runtime and the relation between them. As shown in figure 1, UI adaptation is seen as a state machine. States represents UI states and transitions describe the transitions from a source state to a target state.

For states, the model defines three types of UI states: 1) the “default state” is the running state (the UI resulted from the design phase) 2) the “required state” is the UI adapted to context change and 3) the “loading_error” state presents the UI when a loading problem takes place.

A “UIstate” is defined as a set of aspects (“UIaspect”) representing the UI at the current time.

A UI aspect (Pleuss et al., 2012) may be a presentation unit (e.g. window, container), a UI element (e.g. widget), a layout (i.e. the disposition of widgets on the container) or a visual appearances property (e.g. color, sizing).

At the runtime, UI aspects are described using “feature” and their values are described using feature’s “attribute”. When an adaptation occurs, the adaptation mechanism updates the “state” of features (i.e. features which define the new UI are selected and others which don’t define the new UI are deselected). After feature’s state update, the

new UI is recomposed using "UIModel" which correspond to the selected features.

From another hand, "transition" represents the transition from one state to another state. A transition is defined as a set of adaptation rules.

Adaptation rules are mainly the "context_aspect constraint" (which define the link between context features and UI features), complemented by "aspect constraint" (describing the link between UI aspects and alternatively called constraint propagation rules (Czarnecki et Kim, 2005)). Adaptations rules are described as an Event-Condition-Action (ECA) rules. The "event" may be a context change or an aspect change. If the "event" is a "context change", then the rule is a "context-aspect" rule. Else, if the "event" is an "aspect change", then the rule is an aspect rule.

Following an "event", a "condition" which describes the context value (or the aspect value) is checked, then, if it is valid, the action is executed to aspects to make the required change.

For the context, this later is defined at the runtime using Boolean "variable".

A context may be a sensed context (i.e. acquired from sensors), profiled context (i.e. acquired from the end user) or a derived context (i.e. acquired from another context data).

For enumerative type, we define 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.

4. THE USE CASE

To illustrate our proposition and describe how the adaptation meta-model will be used, we present in this section the "search for restaurant" use case. This use case highlights the adaptation of the main interface of the application to the user preferences change.

User preference is a profiled context information, provided by the end-user and which may address the customization of two main UI aspects:

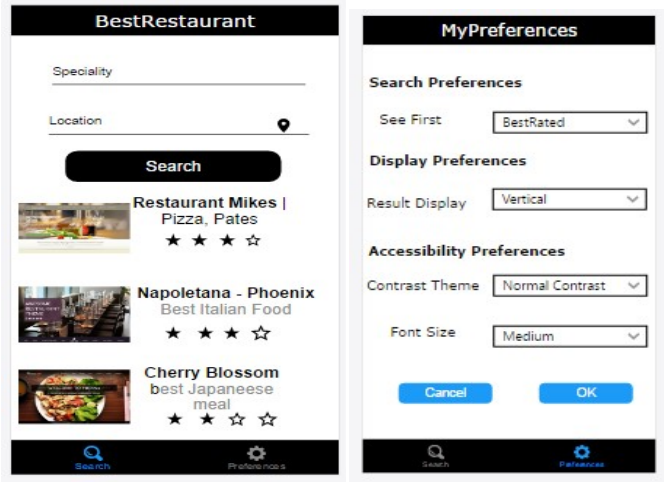
- The presentation aspect: customize the UI structure (UI elements, presentation unit), the UI layout, or the visual appearance of the interface;
- The behavioral aspect: customization of UI element by injecting alternative JavaScript handler.

The application has two interfaces: a search interface and a preferences settings interface.

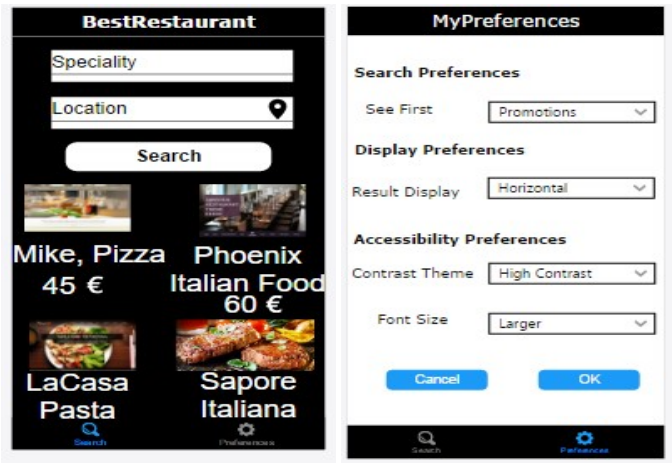
The search interface (figure 2) includes a text field to enter the restaurant speciality, a text field to enter the current location and a search button to validate the search request. By default, search results are displayed as hyperlinks (image hyperlink and text hyperlink) describing the restaurants which correspond to the search request.

The preference UI (figure 2) includes "search preferences", "display preferences" and "accessibility preferences". The "Search preferences" address the customization of the behavioral aspects of the search UI while "display preferences" and "accessibility preferences" address the customization of the presentation aspect of the search UI.

To customize the behaviour of the search UI, "Search preferences" include a combobox specifying the type of restaurant the user is looking for (e.g. best rated restaurant or restaurant offering a promotion). To customize the display of the search result, "Display preference" includes a combobox specifying the type of the preferred display (e.g. a vertical display, or a horizontal display). To make the search interface more readable for user with visual disabilities, "accessibility preferences" define two combobox. The first combox allows the customization of the



(a) Application interfaces before adaptation



(b) Application interfaces after adaptation

Figure 2: The "search for restaurant" use case

contrast theme of the UI and the second combobox allows the customization of the interface.

5. META-MODEL INSTANTIATION

In this section, we instantiate the adaptation model according to the use case described above.

As depicted in figure2, the “search for restaurant” use case defines two states: a default state, conform to the default preferences settings and a required state, conform to the new preferences settings.

In the following, we describe the states of the search UI, adaptation rules allowing states transition and the context of use change.

5.1 User interface states

Figure 3 shows the search UI as it was designed at the design phase while figure 4(a) and figure 4(b) show the search UI states as presented at the runtime phase. At both phases, the search UI is described in terms of features. Features are graphically presented as trees, called a feature diagrams.

At the design phase, the features describing the search UI are defined according to the following aspects:

- UI elements (e.g. “speciality_TextField” feature, “searchButton” feature);
- Presentation unit (e.g. “requestContainer” feature, “ResponseContainer” feature);
- Visual appearance (e.g. “theme color” feature, “Fontsize” feature);
- Layout (e.g. “layout” feature).

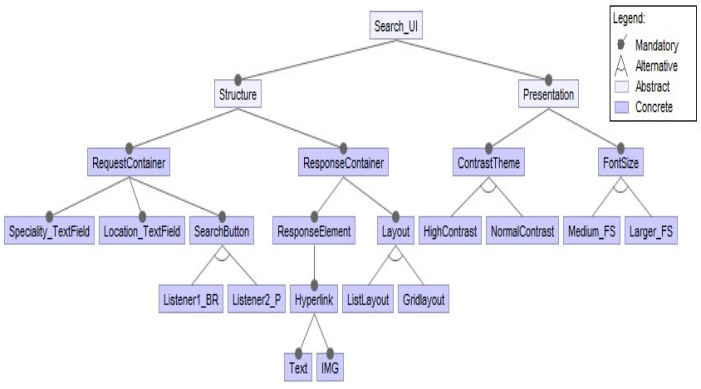
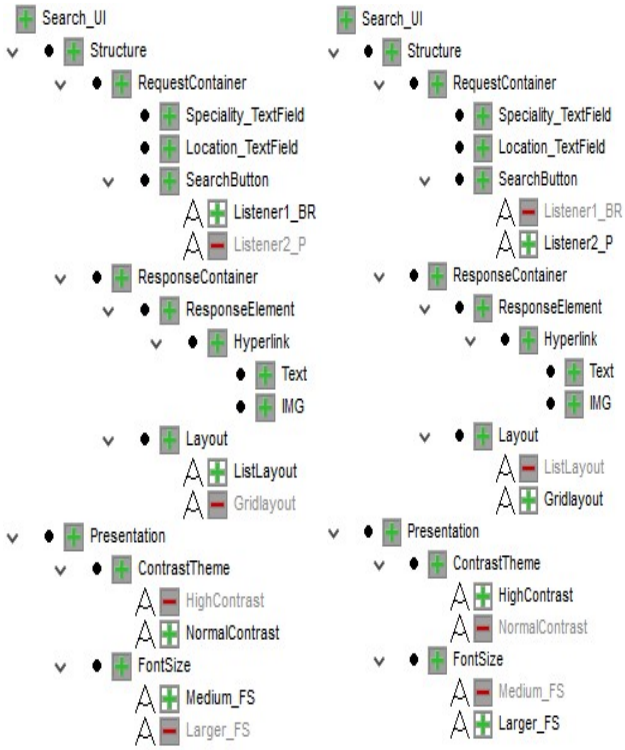


Figure 3: *The search UI feature diagram*

At the runtime, in the feature diagram presenting the default UI state (figure 4 (a)), the “listener_BR” feature is selected to display the best rated restaurant. The “listlayout” feature is selected to display the search result in a vertical way, the “normal contrast” and the “medium” feature are selected to make the interface lighter and the text displayed with a medium size.

After runtime adaptation (figure 4(b)), the“listener_BR” feature is deselected and the “listener_P” feature is selected to display the promotions of restaurants. The “listlayout” feature is deselected and the “Gridlayout” feature is selected to display the result in a horizontal way. The “normalcontrast” and “medium” features are deselected to select “contrasttheme” and “larger” features to make the UI darker and the text size larger.



(a) The default state (b) The required state

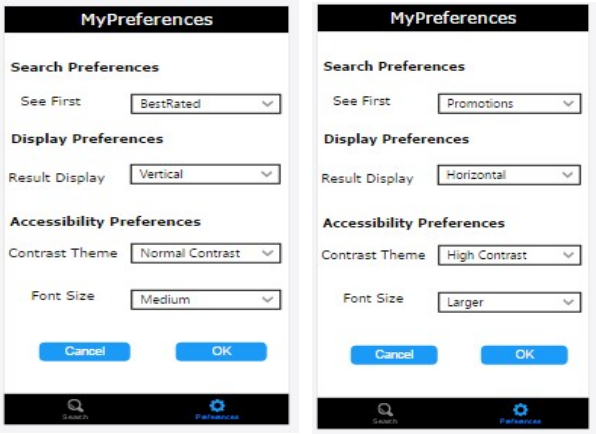
Figure 4: The Search UI states at the runtime

5.2 The context of use

A UI adaptation is triggered following a context change. In this section, we describe the change between two contexts of use.
Figure 5 (a) and figure 5 (b) describe the context change as managed at the runtime phase.

The context of use as depicted in figure 6(a) presents the default user preferences. These preferences are relative to “User1”. “User1” prefers visualizing the “bestrated” restaurants displayed “vertically” and in a “normal” contrast theme and a “medium” font size.

Figure 6 (b) depicts the user preferences ative to "User2". User 2 is a visually disabled user who prefers visualizing the "promotions" of restaurant displayed "horizontally" in a "high" contrast theme and a "larger" font size.



(a)The Default context of use (b) The New context of use

Figure 5: The context of use change

As shown in the model, the context of use is presented at the runtime phase as Boolean variables. Figure 6 (a) and figure 6 (b) describes respectively, the default context of use of figure 6 (a) and the new context of use of figure 6 (b) using Boolean variables.

BestRated	True
Promotions	False
Vertical	True
Horizontal	False
NormalContrast	True
HighContrast	False
Medium	True
Large	False

BestRated	False
Promotions	True
Vertical	False
Horizontal	True
NormalContrast	False
HighContrast	True
Medium	False
Large	True

(a) Default context data (b) the context of use change

Figure 7: the context of use change at the runtime

5.3 Adaptation rules

Table 2 shows adaptations rules allowing the transition from the default state of the search UI to the required state.
According to the use case, we have only context-aspect constraints which describe the link between context variables and UI features.

Table2: Adaptation Rules

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

For example, "AR1" means that if the user select the "bestrated" search preferences value, the "listener1_BR" will be selected to display the best rated restaurant that conform to the speciality and the location entred by the user.

"AR4" means if the "horizontal" display is selected, then the "layout" which will be selected to display the search result in the search UI is a "gridlayout".

CONCLUSION

In this paper, we have proposed a meta-model which describes the runtime adaptation in a UI-DSPL process. To describe how the meta-model may be used, we instantiate it according to an illustrative use case. In the future work and in order to validate the meta-model, we will implement the runtime adaptation mechanism according to the proposed meta-model.

REFERENCES

Boucher Q., et al., "Deriving configuration interfaces from feature models: A vision paper". In Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems (pp. 37-44). ACM, January 2012.

Capilla R.et al., "An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry". Journal of Systems and Software, 2014

Czarnecki K. , Kim C. H. P., "Cardinality-based feature modeling and constraints: A progress report". In International Workshop on Software Factories (pp. 16-20), october 2005

Gabillon Y., Biri N. and Otjacques B., "Designing an adaptive user interface according to software product line engineering". Proc. ACHI, 15, 86-91. 2015.

Garces K., et al., "Variability management in a model-driven software product line," Rev. Av. en Sist. e Informatica,vol.4,no.2,2007

Gomaa H., and Hussein M., “Dynamic software reconfiguration in software product families”. In PFE, pages 435–444, 2003

Kramer, D. M. (2014). “Unified gui adaptation in dynamic software product lines”(Doctoral dissertation, University of West London).

Mostefaoui G. K., Pasquier-Rocha J. and Brezillon P., “Context-aware computing: a guide for the pervasive computing community”. In Pervasive Services, IEEE/ACS International Conference on(pp. 39-48). IEEE, 2004.

Quinton C., Mosser S., Parra C., Duchien L., “Using Multiple Feature Models to Design Applications for Mobile Phones”. MAPLE / SCALE workshop, colocated with SPLC’11, Aug 2011, Munich, Germany. pp.1-8, 2011.

Pleuss A., Wollny S., & Botterweck G., “Model-driven development and evolution of customized user interfaces”. In Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems (pp. 13-22). ACM, June 2013.

Pleuss A., et al., “A case study on variability in user interfaces”, Proceedings of the 16th International Software Product Line Conference - Volume 1, SPLC ’12, ACM, New York, NY, USA, pp. 6–10, 2012.

Pohl K. et al., “Software Product Line Engineering”. Springer, 2005

Schmidt D. C., “Model-driven engineering”. COMPUTER-IEEE COMPUTER SOCIETY-, 39(2), 25, 2006

Gonzales-caleros J. M. et al., https://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/, 2010