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

Thouraya Sboui, Defended Saturday, Pr Abdelmajid, Ben Hamadou, Pr Jean Vanderdonckt, Pr Adel, M Alimi, Pr Mounir, Ben Ayed

Tunisian Rupublic
Ministry of Higher Education and Scientific Research

University of Sfax
National School f Enginnering of Sfax

Doctoral School of Sciences and Technologies

PhD Thesis

ISI
N° 1250

THESIS

Submitted and publicly defended in partial fulfillment of the requirements for the Award of Doctor of Philosophy in

Computer Systems Engineering

Presented By

Thouraya SBOUI

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

Pr.	**Abdelmajid Ben Hamadou**	ISIM, University of Sfax	President
Pr.	**Jean Vanderdonckt**	UCL, University of Louvain	Reader
Pr.	**Adel Mahfoudhi**	CCIT, Taif University	Reader
Pr.	**Nadia Bouassida**	ISIM, University of Sfax	Examiner
Pr.	**Adel M. Alimi**	ENIS, University of Sfax	Thesis Director
Pr.	**Mounir Ben Ayed**	FSS, University of Sfax	Advisor

Defended Saturday, March 31, 2018

11:00 Am

Abstract

In the modern world of mobile computing and ubiquitous technology, society is able to interact with technology in new and fascinating ways. To help provide a permanent service, mobile software should be adapted to suit the user preferences. By monitoring context information relative to the final user, the application can better meet the dynamic preferences of the user. This program commonality and variability can benefit from the use of Software Product Line Engineering, reusing artefacts over a set of similar programs, called a Software Product Line (SPL). Historically, SPLs are limited to handling static compile time adaptations. Dynamic Software Product Lines (DSPL) however, allows for the program configuration changing at runtime, allowing for compile time and runtime adaptation to be developed in a single unified approach. While currently DSPLs provide methods for dealing with program logic adaptations, variability in User Interfaces (UIs) has largely been neglected. Due to this, depending on the intended time to apply UI adaptation, different approaches are required. The main goal of this work is to propose an SPL approach for the development context-adaptable UIs. As context element, we choose to adapt our user interfaces to the user preferences. Our approach is intended to handle UI adaptation within DSPLs, providing a unified representation of UI variability is presented. The approach is based on Model Based User Interface Development Models, enabling developers to implement UI and context variability. To validate our approach, we implemented a design phase and a runtime phase prototypes according to a proposed illustrative example.

Résumé

Dans le monde moderne de l'informatique mobile et de la technologie omniprésente, les utilisateurs sont capables d'interagir avec la technologie de manière nouvelle et fascinante. Pour aider à fournir un service permanent, les logiciels mobiles doivent être adaptés au contexte d'utilisation entre autres les préférences de l'utilisateur. En tenant compte du contexte d'utilisation, l'application peut mieux répondre aux préférences dynamiques de l'utilisateur. L'uniformité et la variabilité de ce programme peuvent bénéficier de l'utilisation de Software de l'ingénierie des les lignes de produits logicielles, réutilisant des artefacts sur un ensemble de programmes similaires, appelé Ligne de produits logiciels (SPL). Historiquement, les lignes de produits sont limitées à la gestion des adaptations du temps de compilation statique. Les lignes de produits logicielles dynamique (DSPL) permet néanmoins de changer la configuration du programme au moment de l'exécution, de permettre au temps de compilation et à l'adaptation de l'exécution d'être développés dans une seule approche unifiée. Alors qu'actuellement, les DSPL fournissent des méthodes pour traiter les adaptations logicielles du programme, la variabilité des interfaces utilisateur (UI) a été largement négligée. En raison de cela, selon le moment prévu pour appliquer l'adaptation de l'interface utilisateur, différentes approches sont nécessaires. L'objectif principal de ce travail est de proposer une approche SPL pour les UIs adaptées au contexte de développement. En tant qu'élément de contexte, nous choisissons d'adapter nos interfaces utilisateur aux préférences de l'utilisateur. Notre approche est destinée à gérer l'adaptation de

l'interface utilisateur dans les DSPL, en fournissant une représentation non modifiée de la variabilité de l'interface utilisateur. L'approche est basée sur des modèles de développement d'interface utilisateur basés sur un modèle, permettant aux développeurs de mettre en œuvre l'interface utilisateur et la variabilité du contexte. Pour valider notre approche, nous avons mis en place une phase de conception et un prototype de phase d'exécution selon un exemple illustratif propose.

Acknowledgements

I have to confess, when I embarked on this PhD, I was only interested in what I felt was the goal, and how best to get it. On this journey, as all great journeys, important people always help shape it. For this, I feel deep gratitude is required.

Firstly, I wish to thank Pr. Mounir Ben Ayed for all the opportunities, guidance, and continual useful critiques of my research.

A big thanks to Pr. Jean Venderdonckt, your help, advice, suggestions and positivity have been important. I, also, thank you for your proposal to become the reporter of my thesis.

Big thanks to Pr. Rafael Capilla for his support, and encouragement in difficult times.

I would also like to express my thanks to Pr. Adel M. Alimi for having accepting me in the REGIM Laboratory and for his guidance.

Many thanks to Pr. Adel Mahfoudhi for being the reporter of my thesis.

Big thanks to my family for their support and their presence during the good times, and the bad. Thank you for making this journey more fun, and in many cases, bearable. I would like to thank my parents, for their sacrifices and selfless love. My sister, Amani, for her constant words of encouragement, and for always being a great sister. For my brothers Rafik, Amine and Wassim for their love, support, and always being there for me. Lastly, I would like to give a special thanks to my Mum for everything she ever did for me. I really could not have asked for a better Mother ♥.

Declaration of Authorship

I, Thouraya SBOUI, declare that this thesis titled, “A UI-DSPL Approach for the development of Context Adaptable User Interfaces” and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a Phd degree at the ENIS University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at the ENIS University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Thouraya SBOUI

_____

Date: 10/03/2018

_____

"It is through science that we prove, but through intuition that we discover"

Henri Poincare

Contents

List of figures

List of tables

Abbreviations

Acronym	Explanation
ACM	Association of Computing Machinery
AIU	Abstract Interaction Unit
AIO	Abstract Interactive Object
API	Application Programming Interface
BDD	Binary Decision Diagrams
CAUI	Context Adaptable User Interface Approach
CVL	Common Variability Language
AUI	AbStract User Interface Model
CIM	Computation Independent Model
CIU	Concrete Interaction Unit
CRF	Cameleon Reference Framework
CTT	Concurr Task Model
CUI	Concrete User Interface Models
CSUQ	Computer System Usability Questionnaire
DBU	Dynamic Bunding Unit
DSL	Domain Specific Language
DSPL	Dynamic Software Product Line
ECA	Event-Condition-Action
EMF	Eclipse Modeling Framework
FAMILIAR	FeAture Model scrIpt Language for manIpulation and Automatic Reasoning
FD	Feature Diagram
FM	Feature Model
FMP	Feature Modeling Plugin
FODA	Feature Oriented Domain Analysis
FOSD	Feature Oriented Software Development
FUI	Final User Interface Model
GPS	Global Positionning System
GUI	Graphical User Interface
HCI	Human Computer Interaction
HTML	HyperText Markup Language
IBM	International Business Machines
IDE	Integrated Development Environment
IFML	Interaction Flow Modeling Language

MBUID	Model Based User Interface Development
MD	Model Driven
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MS	MicroSoft
OMG	Object Management Group
OOP	Object Oriented Programming
PIM	Platform Independent Model
QVT	Querry, view, Task Language
SAT	SATisfiability Solvers
SMV	Symbolic Model Verifier
SPEM	Software Process Engineering Meta-Model
SPL	Software Product Line
SPLC	Software Product Line Conference
SPLOT	Software Product Line Online Tools
SPLE	Software Product Line Engineering
UsiXML	User Interface eXtended Markup Language
VaMos	Variability Modelling of Software-Intensive Systems
XML	eXtensible Metadata Language
XSLT	eXtensible Stylesheet Language Transformation

Chapter 1. Introduction

1.1 Introduction

In this chapter, we state the motivation of our thesis, the problem that must be resolved, the planned solution, the main concepts of our dissertation, our publications during years of work and finally the thesis roadmap. In the problem statement, we present the problem, the research questions which guided our research and the proposed solution. Furthermore, we present the most important concepts related to our research areas. Among the concepts, we find: user interface variability, the context of use, UI adaptation and software engineering approaches used for user interface development. Finally we enumerate our publications during the thesis.

1.2 Motivation

The rapid growth of computing devices, the diversification of their context of use, and the variety of user profiles are creating competitive challenges. Such a progress seems promising for the user interface (UI) field to offer personalized interfaces and interaction scenario that correspond to user expectations. UI adaptation is an active domain of research in Humain Computer Interaction (HCI). For this reason, adaptation approaches are evolving with the context of use in order to increase user's satisfaction and enhance interaction experience. Adaptation methods are evolving to fulfill new requirements and increase UI efficiency. By attempting to cut with earlier interfaces that often needed recompilation for upgrades, which incurred increased cost, delay, and risk, UIs

shift to a runtime paradigm. UIs turn out to be adaptive rather than being user-centered and carry out adaptation in accordance with the end-user preferences as well as the context of use.

1.3 Problem statement

The problem of our thesis is how to tackle to the diversification of user preferences and develop a family of context-adaptable user interfaces. To meet this need, many approaches opted for the use of generative approaches of software engineering. Among these approaches, many had used the Software Product Line paradigm in order to develop a family of usable interfaces [26][28][31][40-42][48][53-56][58-59][66][71]. Software product lines refers to software engineering methods, tools and techniques for creating a collection of similar software systems from a shared set of software assets using a common means of production. Carnegie Mellon Software Engineering Institute defines a software product line as "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". In the following, we will express the research questions which guided our thesis and to which we have to answer.

1.3.1 Research questions

Based on the aim of this research, this thesis attempts to answer the following research questions:

1) **Which context element will be addressed**? This question is addressed in both chapters 4 and 5. Contrary to existing approaches, we will address the user element, in particular, his preferences vis-à-vis his user interface. User preferences are a context data that have to be manually entered by the end-user. In that case, the context is called a “profiled context”.

2) **How design and manage the context of use**? This question is addressed in both chapter 4 and chapter 5. In the design phase, the context is designed and implemented in order to generate a context interface (to set the user preferences) and in the runtime phase, the context is manually provided by the user through the context interface and automatically managed by the runtime adaptation mechanism;
3) **How to make the UI-DSPL approach more abstract**? This challenge will be addressed in chapter 4. At the design phase and to implement feature models, we will use as implementation artifact, the MBUID models. The use of models, instead of, component or aspect, makes the UI-DSPL approach more abstract.
4) **How to support the design and the development of the runtime adaptation mechanism?** This question is addressed at the design phase (chapter4) by proposing a design pattern, called a runtime adaptation model.

1.3.2 Adopted Solution

Unlike the SPL paradigm, the DSPL [14][22][31] paradigm continues to configure and adapt at the runtime. The DSPL provides a unified solution to tackle the need for both design-time and runtime adaptation. To answer to the research questions, we propose, in this dissertation, an approach that supports the complete software life cycle: from feature selection and initial product derivation, to runtime adaptation in response to changes of user preferences. We concretize the notion of asset with a definition of models to realize the variability across a family of products. In our approach, the derivation of products is divided in two processes: a design time process for the generation of default user interfaces and a runtime process for the adaptation of UIs.

1.4 Main concepts

The most important concepts relative to our work are: 1) user interface variability [54], a main concept of the development of a family of UIs using the SPL paradigm, 2) the context of use, a main concept of the development of context-aware systems, 3) UI adaptation: this concept highlights different type of user interface adaptation (e.g. automatic adaptation, manual adaptation and semi-automatic adaptation) and finally the 4) generative approaches, this later highlights different types of generative approaches used to develop user interfaces. In the following, we will detail these concepts.

1.4.1 User interface variability

Like the other aspects, user interface variability may be designed and implemented. This variability is defined according to many aspects such as the presentation aspect and/or the functional aspect of a user interface. Current SPL techniques and tools are dealing with this variability at the design time. However, managing the runtime variability in order to adapt the UI cannot be achieved using a conventional SPL approach. Runtime variability can only be tackled using Dynamic Software Product Line techniques. This variability can be expressed in stand-alone models, such as feature models (FM) [4] and implemented using different artifacts (e.g. models, component, document, etc…).

1.4.2 The Context of use

The context of use was defined by many teams [12] [26] [35] [40] [70] by the triplet <user, platform, environment>. Context elements can be defined as follows:

- The platform element presents any information pertaining to the software or the hardware platform (processor, memory, peripheral equipments, connection network, the size of the display screen, the available interaction tools, etc…);
- The user element presents any information relative to the user (e.g. profile, his current activity, preferences, habits, cultural characteristics, etc…);
- The information corresponding to the environment (light, noise, geographical localization, etc …).

In some proposals [53] [55], this definition was extended by adding another element like the customer element. In addition to the end user, the customer is the person who owns the concrete product. Often, this is not the end user itself. In other proposals, the context has been defined differently. For example and according to Mostafeoui [47], three types of context were identified. We find the Sensed context, the derived context and the profiled context:

- Profiled context: refers to the context that the user provides explicitly (for example, the entries in the user profile);
- Sensed context: this context is acquired from the environment by means of physical or software sensors (identity, location, temperature, time);
- Derived context: this kind of contextual information is inferred from another context information (e.g. from Profiled and/or Sensed) using some derivation process. For example, the name of the city from GPS coordinates through a relation mechanism.

To develop a context-aware user interfaces, the context may be considered at the design phase [12][40][41][53][55][70], at the runtime phase [12][26][40][41], or at both phases [12][19][26]. This consideration requires the design of the context. The context was designed in several approaches using models (e.g. user model, platform model, environment model). These models are used in design phase and in runtime phase [12].

1.4.3 User interface Adaptation

Context-awareness and user interface adaptability are reciprocally interrelated. In context-aware or adaptable UIs, the context of use is designed in order to adapt user interfaces to context changes. In the following, we present three types [13] of user interface adaptation:

1.4.3.1 Adaptive UI

An Adaptive UI refers to a UI capable of being aware of the context of use and to (automatically) react to changes of this context in a continuous way (for instance, by changing the UI presentation, contents, navigation or even behaviour).

1.4.3.2 Adaptable UI

An Adaptable UI can be tailored according to a set of predefined options. Adaptability normally requires an explicit human intervention. We can find examples of UI adaptability on those word processors where the set of buttons contained by toolbars can be customized by end users.

1.4.3.3 Plastic UI

A Plastic UI is a multi-target UI that preserves usability across multiple targets. Usability is not intrinsic to a system. Usability can only be validated against a set of properties set up in the early phases of the development process.

1.4.4 Generative software engineering approaches and UI adaptation

Software Engineering is still struggling to produce a family of software systems, amongst these approaches, we find, mainly, the Model Driven Engineering approach (MDE) [3][6][28][67], the Model Based User Interface Development (MBUID)[12] [13][32][45][70][74] approach and the Software Product Line Engineering (SPL) approach.

1.4.4.1 Model Driven Approaches

Model Driven Approaches are approaches which use the Model Driven Engineering (MDE) [3][6][28][67] paradigm (figure1-1). The MDE paradigm is a software development methodology which considers the model as a central artifact leading to the design of the system and the generation of the desired software. One of the key points of MDE is the raise of abstractions [67]. MDE defines four levels of abstraction: the meta-meta-model, the meta-model, the model and the code. These levels are connected by means of transformations. Within the field of UI adaptation, an overview of model driven user interface approaches was provided by *Akiki et al.* [3].

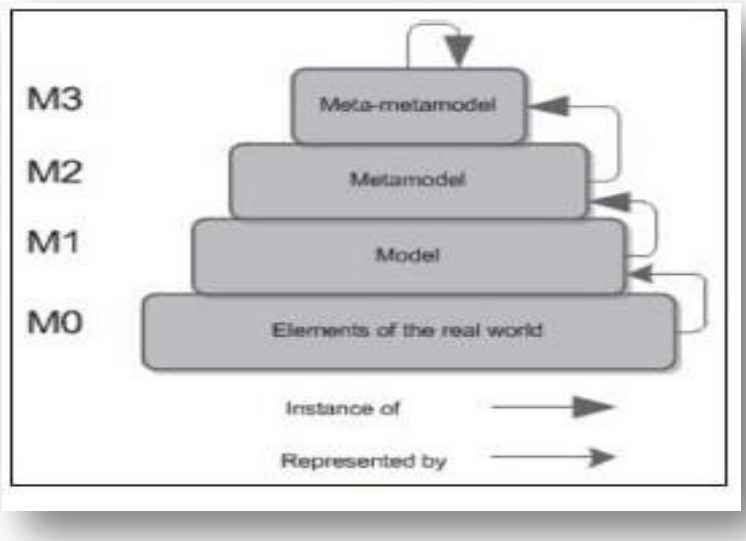


Figure 1-0-1 The four layer architecture of MDE

1.4.4.2 Model Based User Interface Approaches

Another important paradigm is the Model Based User Interface Development (MBUID) [12] [13][32][45][70][74] paradigm. This paradigm is a particular MDE process defining a specific user interfaces models.

Most MBUID approaches rely on the Cameleon Reference Framework (CRF) [12]. The CRF serves as a reference for the development of UIs supporting multiple targets or multiple contexts of use.

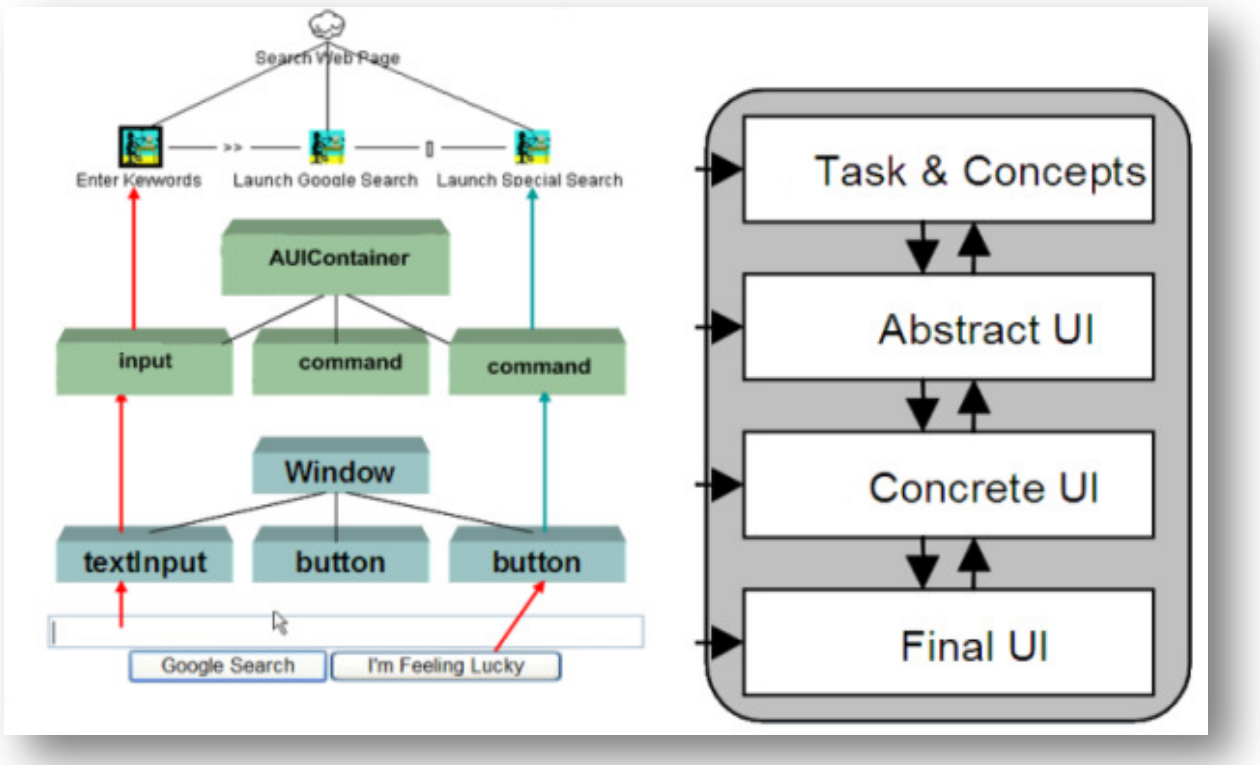


Figure 1-0-2 An instantiation of the CAMELEON Reference Framework [12]

The CRF [13] is illustrated in figures 1-2 and 1-3. Within this platform there are both task and domain models. The task model describes the logical activities that have to be carried out in order

to reach the user's goals while the domain model is a conceptual model of the domain that incorporates both behavior and data. Both models are used as input to generate the abstract user interface (AUI) which expresses the UI in terms of Abstract Interaction Units (AIU) (or Abstract Interaction Objects (AIO)), as well as the relationships among them. These AIUs are independent of any implementation technology or modality (e.g. graphical, vocal or gestural). The AUI may give rise to one or many Concrete User Interface (CUI). A CUI expresses the UI in terms of Concrete Interaction Units (CIU) (or Concrete Interaction Objects (CIOs). These CIUs are modality-dependent but implementation technology independent. The CUI is used to derivate the Final Interactive UI (FUI).

In the context of user interface adaptation, the CRF considers the context of use at the design time and at the runtime in order to support plastic UIs. This consideration is performed using context models (user model, platform model, environment model). Context models are used to ensure the transformations between models of the four abstraction levels.

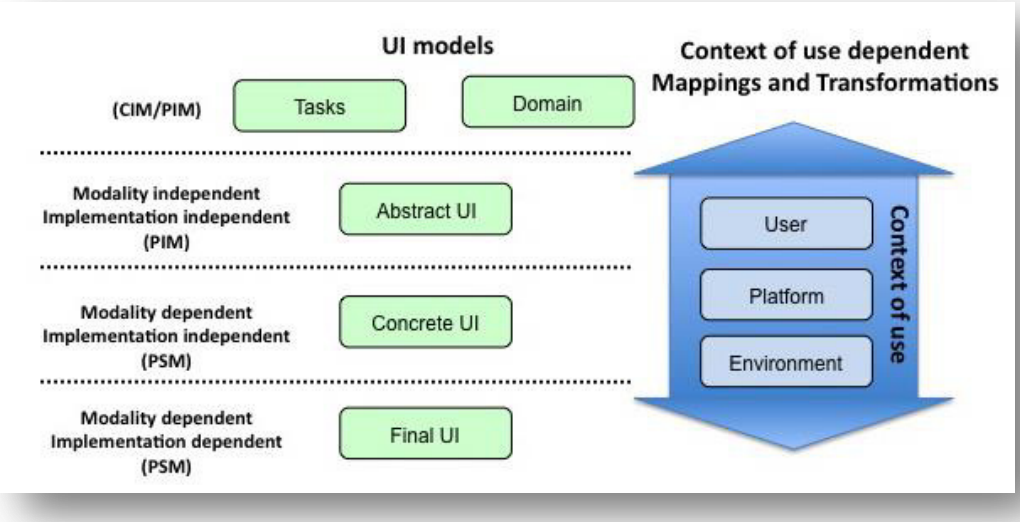


Figure 1-0-3. A simplified version of the Cameleon Reference Framework (CRF). Mappings and transformations between levels of abstraction depend on the context of use.

1.4.4.3 Software Product Line Approaches

The third generative process is the Software Product Line Engineering (SPLE) process [26][28] [31][40-42][48][53-56] [58-59] [66] [71][76]. SPLE promises significant improvements in time-to-market, cost, and reliability through the system identification and the exploitation of commonalities and variations in software systems.

SPLE is a two-level approach (Figure 1-4): an abstract level called the domain engineering level and a concrete level called the application engineering level. The domain engineering level covers domain analysis (identification of common and variable features among the family members), domain design (development of common assets of all family products) and domain implementation (the implementation of the family-assets). The application engineering level covers application analysis (the configuration of the feature model), application design (instantiation of common assets in order to define the architecture of a specific product) and application implementation (the development of a specific product).

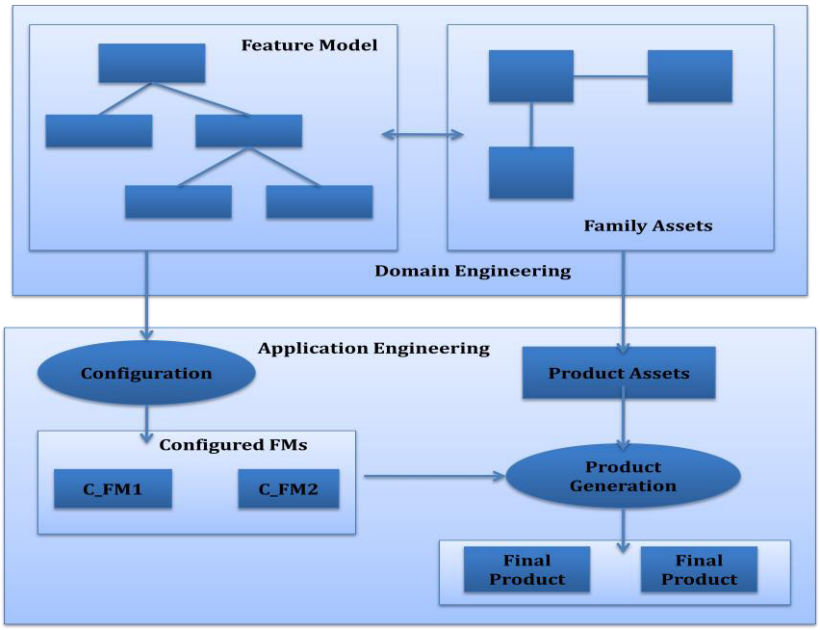


Figure 1-0-4. Software Product Line Process

Both engineering levels define two spaces: a problem space and a solution space. The problem space is for expressing variability and product configuration while the solution space is for assets implementation and product generation. Figure 1-5 highlights concepts and technologies used to implement the two spaces.

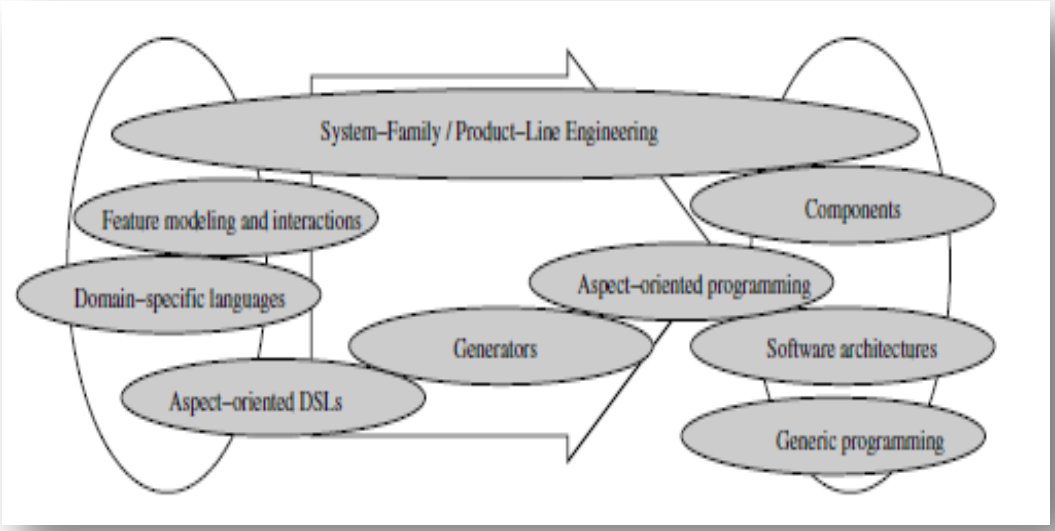


Figure 1-0-5 Relationship between generative software development and other fields [46]

Feature modeling [4] is a key concept in product line engineering. Feature modeling by means of Feature Diagrams (FD) is a popular technique for capturing commonality and variability in Software Product Lines. A feature model is an intermediate step to move from problem space to solution space.

In the context of UI adaptation, many software product line approaches were proposed [30][35][40][43][46]. Some proposals had considering the context of use only at the design time, others, had considered the context at the design time and at the runtime phases. These laters opted

for the use of Dynamic SPLs approaches [26] [31] [40][41]. DSPLs and unlike SPLs continue to adapt software systems at the runtime.

1.5 Publications

1.5.1 Journal [65]

T. SBOUI, M. BEN AYED, A. M. ALIMI, "A UI-DSPL approach for the development of Context-adaptable UIs", IEEE Access, Vol PP, 2017

1.5.2 Journal [61]

T. SBOUI, M. BEN AYED, "Generative Software Development Techniques of User Interface: Survey and Open Issues", IJCSIS Journal, Vol 14 N°7, 2016

1.5.3 International Conference [62]

T. SBOUI, "A DSPL Approach for the Development of Context-Adaptable User Interfaces", RCIS Conference, 2017

1.5.4 International Conference [63]

T. SBOUI, M. BEN AYED, A. M. ALIMI, "A meta-model for run-time adaptation in a UI-DSPL process", BHCI Conference, 2017

1.5.5 Book Chapter [64]

T. SBOUI, M. BEN AYED, A. M. ALIMI, "Addressing Context-Awareness in User Interface Software Product Lines (UI-SPL) Approaches", Human–Computer Interaction Series, 2017, ISSN: 1571-5035.

1.6 Thesis Road Map

Our dissertation is broken down into the following chapters:

• Chapter 2: Software Product Lines for User Interfaces. This chapter introduces existing Software Product Lines approaches proposed in the context of user interface development. In this chapter, we attempt to give a literature review to determine the gaps that have to be fulfilled by our contribution.

• Chapter 3: A UI-DSPL Approach for the development of Context-Adaptable User Interfaces. Based on the identified gaps of existing works, the third chapter presents the contribution of our thesis. We introduce a Dynamic Software Product Line as an approach for developing adaptive softwares, and their properties. Next we present the challenges that fulfill the gaps of existing proposals. Furthermore, we introduce the use case that will illustrate the rest of chapters. Finally, we detail the two phases of the proposed approach. The design time stage includes the variability design, implementation, and derivation. While, the runtime phase presents the runtime configuration and recomposition of the new interface.

• Chapter 4: Implementation. This chapter presents the implementation details of our approach. Among these details, we find the development tools, source codes of the implemented algorithms and the generated prototypes for the mobile platform.

• Chapter 5: Evaluation&Discussion. In this chapter, we evaluate the proposed approach using three types of evaluation: a scenario-based evaluation, a scalability evaluation and a questionnaire-based evaluation.

• Chapter 6: Conclusions and Future Works. This chapter presents the final conclusions of the work featured in this thesis. We lastly present motivated work that should be further researched within the field.

1.7 Conclusion

This chapter had introduced our dissertation by fixing the motivation, the problem for which we are looking for a solution, concepts that are relative to our work, our publications and the roadmap of our thesis. The next chapter presents an overview of software Product line approaches used for user interface adaptation.

Chapter 2. Software Product Lines for User Interfaces

2.1 Introduction

This chapter is a state of the art within it we present software engineering approaches proposed in the context of user interface adaptation. First, we present the list of conference, journals and workshops in which we have conducted our research to find publications that are related to our topic. Second, we present a comparative study of existing SPL approaches reserved for user interface adaptation. Then, we propose a summarizing schema, on which we tag these approaches and which will serve as a design/development pattern for the developers wanting to develop a UI-SPL approach.

2.2 Software product line approaches for user interface development

2.2.1 Resources for Finding and Accessing Scientific Papers

The table below (table2-1) presents the most important workshops, conference, journals of software engineering (in particular software product line engineering, e.g. SPLC, VaMos) and of Human Computer Interaction (e.g. the ACM transaction on computer human interaction) topics. These resources were selected according to their ranks and h-index. The higher the rank, the more selective the source is, the higher the h-index, the better the source is. However, it is possible to

find sources which are considered as having an important venue in product line engineering but which are not ranked (e.g. the SPLC conference).

Table 2-1 List of conferences, workshops

	Acronym	Full Name	Rank/ H-index	Search results
Conference/Workshop	GPCE	Generative Programming conference engineering	B/-	--
	SPLC	Software Product Line	-/-3232	[58] [66] [54]
	VaMos	Variability Modeling of software	-/5	[11]
	ACHI	Advances in Computer Human Interaction	C/8	[26]
	AVI	Advanced Visual Interfaces	B/24	--
Journal	IJHCI	International Journal of Human-Computer Interaction	A/38	--
	TCHI	ACM transactions on computer-human Interaction	B/46	--
	CEJCS	Central European Journal of Computer Science	-/-	[7]

2.2.2 An overview of findings

After presenting the resource of findings, we present in table 2-2, the SPL approaches used for the development of user interfaces. These approaches are compared according to a list of criteria. These criteria are:

- **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 [40][41]);
- **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.

Schlee &Vanderdonckt [66] 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. [28] 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. [58] 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

Table 2-2 An Overview of UI-SPL approaches

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]		+	+			+		+	-	-	+	+	+

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.

Müller [48] 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. [11] 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. In [11], the authors don't generate the final UIs, they only present the interface sketches. Furthermore, there is neither context management nor interface adaptation.

Pleuss et al. [53] [55] propose an approach which includes only a design phase in which the target context element was the customer. In [55], the authors used the Model-Based User Interface Design (MBUID) models 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. These authors also used MBUID models in [55] 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. [6] use a model driven approach based on a decision model to generate 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's 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. [42] 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 [42], there is nor context management neither UI adaptation.

Kramer [40] [41] 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. [26] 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. [71] 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. In [71], Sottet doesn't 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 [12][53][70][74] 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.

Calvary et al. [12] 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 et al. deals with a particular interface adaptation which is UI plasticity. This latter is defined by Calvary et al. as "the capacity of user interfaces to adapt to the context of use while preserving human values".

Sottet et al. [70] 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.

Mezhoudi et al. [46] 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 et al.'s approach is among the first MBUID approaches that have used user feedbacks to adapt user interfaces.

UsiXml [74][75] 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. [27] 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 et al.'s approach is that is not dedicated for the development of a family of UIs.

Cerny et al. [16] 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

2.3 Summary Schema

In order to summarize the approaches described above, we designed a schema which will serve as a reference for the designers who want to develop a MB-SPL approach to develop context

sensitive user interfaces. The reference schema combines different type of software engineering artifacts in order to develop a UI-SPL process. The global process is a two-layered SPL process presenting a domain engineering level and an application engineering level in which they aggregate SPL artifacts (e.g. feature model, feature configuration), MDE/MBUID artifacts (e.g. model, metamodel, transformation) and context consideration techniques (e.g. context feature model, runtime adaptation mechanism) made to the development of a family of context-.

2.3.1 Design Elements

In figure2-1, the design elements are logically distributed throughout the schema levels. The domain analysis level is the process of identifying, eliciting and modeling the requirements of a family of products. The major design elements that can be included in this phase are:

- Metamodels: defines the metadata of SPL/MDE artifacts. We find metamodels describing variability models, metamodels describing MBUID models, and metamodels describing MDE models. Another metamodel can take place is that which describes the runtime adaptation mechanism, this metamodel serves to facilitate the design and the development of the runtime adaptation mechanism;
- Variability models: defines the commonalities and the variabilities of the user interface and the context of use. To define variability models, we use Feature Diagrams (FD);
- UI domain model: defines the meaningful real-world concepts pertinent to user interface domain. It may be presented by a UML class diagram;
- Task model/Interaction Flow Modeling Language (IFML) model [49]: the task model is the correspondent of the use case diagram in UML language and it represents the logical activities that should support users, interacting with the interface, in reaching their goals.

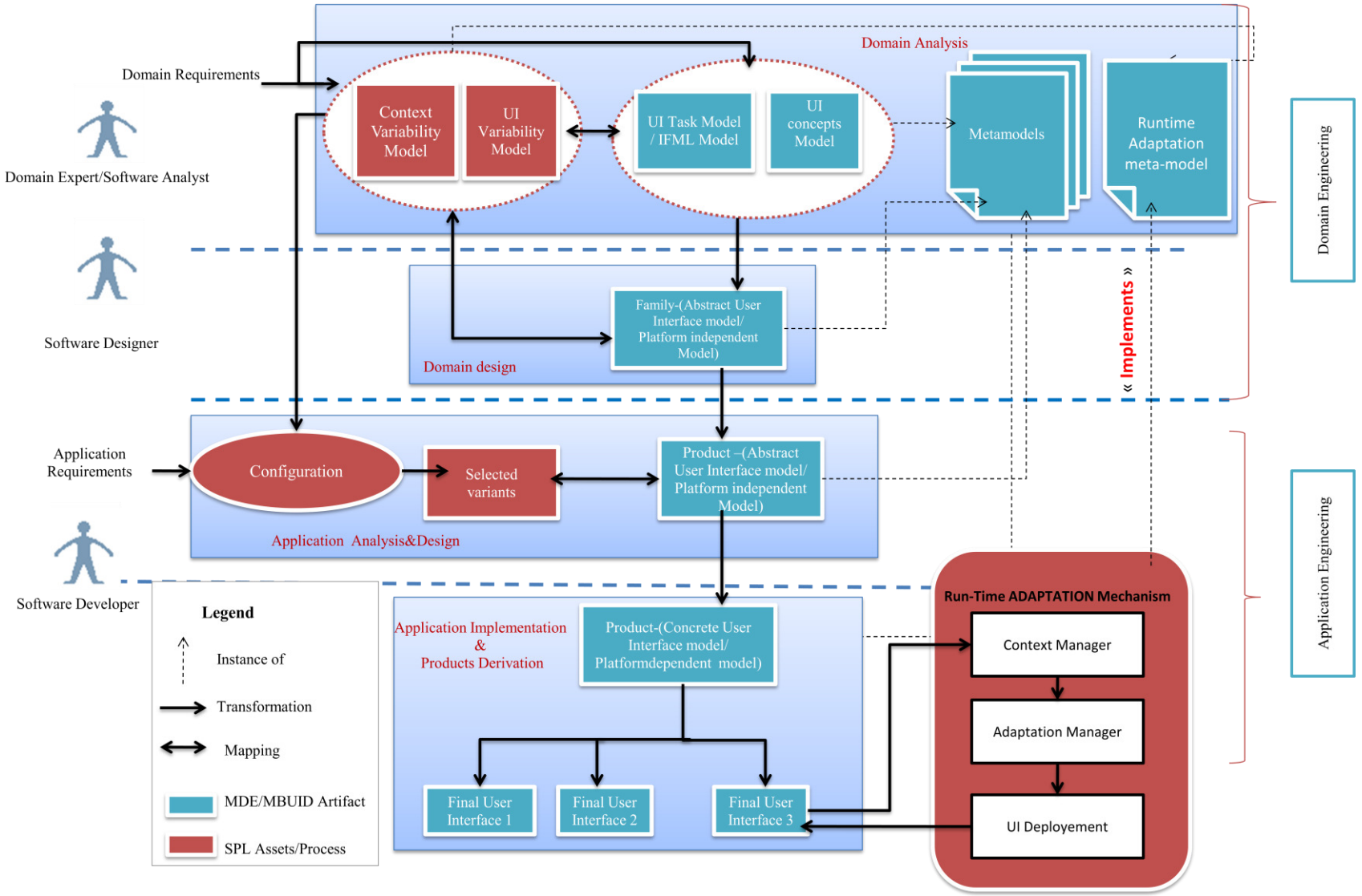


Figure 2-1 A summay/reference schema for the development of user interfaces using SPL and MDE core assets

The Concur Task Trees (CTT) [52] [60] is a visual notation used to describe the task model. Regarding the IFML [49] model and beyond the description of user interactions, the IFML is designed for expressing content, control behaviour of the front-end of software applications, as well as the binding to the persistence and business logic layers. The IFML is a Domain Specific Language (DSL) that has been adopted as a standard by the Object Management Group (OMG) in March 2013;

The domain design level takes domain models described above and aims to produce a generic architecture to which all UIs can conform. This architecture is described using the family specific Abstract User Interface (AUI) model or the family specific Platform Independent Model (PIM).

- Family-specific AUI/PIM models: both models describe the UI family in terms of interaction spaces (or presentation units), independently of which interactors are available and even independently of the modality of interaction. These models are designed at the domain engineering level to describe presentations units composing the whole family of UIs;

The application engineering level is characterized by the derivation of the user interface product. This derivation satisfies specific application requirements. At this level, we find the following elements:

- Selected variants: specifies selected and deselected variants of UI and context variability models. At this phase, selected context variants as well as specific application requirements impacts the configuration of the variability model of the user interface;
- Product-specific AUI/PIM models: these models are an instantiation of the family-specific AUI/PIM. At the application engineering level, these models describe a

specific UI in terms of interaction spaces and independently of which interactors are available and the modality of interaction. From another hand, these models present assets used to implement the selected variants of the UI variability model.

- Product-specific Concrete user interface model/platform specific model: these models describe the interface in terms of concrete interactors that depend on the used modality. These models are specific to a particular UI product and are generated from the product specific AUI/PIM model thanks to an MDE transformation.
- Final user interface: depending on the target platform, this model specifies the source code of the UI in any programming language or mark-up language. The source code can be interpreted or compiled.

The application engineering level defines three types of processes:

- The configuration process: is the customization of the variability models by selecting and deselecting the appropriate variants in order to meet specific user requirements;
- The runtime adaptation mechanism: this process is responsible of UI adaptation to context changes when the UI is running;
- Transformation: is the connection linking design elements of the different levels of the development process (more details about transformations are given in the next section).

2.3.2 Transformations

The summary schema defines four types of transformations connecting design elements of higher level of abstractions to design elements of lower level of abstractions. These transformations can be automated (performed by the computer autonomously), semi-automated (requiring human

intervention) or can be performed a manually. The different transformations defined by the summary schema are:

- Instantiation: this transformation specifies the metamodels to which SPL and MDE models must conform. An instantiation is an automated transformation which may be performed using Integrated Development Environments (IDE) (e.g. eclipseIDE, featureIDE);
- MDE/MBUID connections: the transformation which connects MDE/MBUID models is an automated way that transforms a source model to a target model or to text (e.g. source code). This kind of transformation is defined using transformation languages, collectively known as QVT (Query/View/transformation) languages [50];
- SPL connections: include connections between the variability models, connections between variability models and their configurations, and connections between features and their artifacts.

  The connection between variability models is supported by a set of composition and decomposition operators (e.g. aggregate, merge, slice) [2].

  The connection between the variability model and its configuration need the user intervention, so they are performed semi automatically using variability modeling IDEs (e.g. featureIDE);
- SPL/MDE connection: is the mapping linking SPL and MDE artifacts. A first mapping links variability models and task/domain models. That means, if we already designed variability models, this will help designers to model the task/domain model and vice-versa. The two other mappings connect variability models to AUI/PIM models, this mapping presents feature transformation into artifacts.

2.3.3 Actors

The common process supports four types of actors, they include:

- Domain Expert: He has a deep knowledge of the domain. After expertise training, the domain expert delivers the glossary of UI terms to the software analyst;
- Software Analyst: the analyst studies the domain knowledge provided by the domain expert, and identifies the functional and non-functional specifications based on user requirements;
- Software Designer/developer: with the collaboration of the software analyst and the domain expert, software designer and developer are responsible of the elaboration of domain analysis models/metamodels and of the family-specific AUI/PIM model. Moreover, software designer/developer are responsible of product derivation (from product specific AUI/PIM) until FUI model;
- Final Users: these are the people who have a stake or interest in the use of interactive systems. They are invited by the analyst to specify their requirements and they are represented, on the schema, by their requirements (domain requirements/application requirements).

2.4 UI-SPL approaches positioning

In this section, the common process described above is used to position the UI-SPL proposals of Table 2-2. This positioning allows knowing the levels/artifacts of the summary schema which have been used most and others levels/artifacts that were ignored. This positioning will help us to see the difference between approaches and to better define the contribution of our thesis.

In need of clarity, the positioning of the approaches is done using three schemas. In Figure 2-2, the approaches [11][48][66][71] were tagged, while in figure2-3, we tag the approaches [28][42][58] and in figure 2-4, we tag the approaches [6][26][40][41]. In the following, we describe the position of each proposal. For each approach, we select the correspondent design elements.

In [71], Sottet et al. use the MBUID core assets. The used design elements were: the "domain model" which corresponds to the "UI concepts model" (in Figure 2-2), the "IFML model" (instead of the task model), the CUI model to generate the concrete UI, and the ISM to generate the final user interface. Concerning variability models, authors use a multiple feature models (to describe the various facets of UI variability).

Boucher et al. [11], proposed an MBUID process to manage variability and generate a configuration UI. To reach this goal, they use the variability model to manage the variability of interfaces, the AUI model to implement the feature model and a CUI model to generate a concrete UI.

Pleuss [53][55] and Muller [48] use the feature model, the task model and the AUI model to define the domain engineering level. Then, they instantiates these models to define the product architecture which corresponds to the CUI model.

In [66], schelee and Vandendanckont use the variability model and the frame technology (which is absent as a design artifact), to generate the final UI.

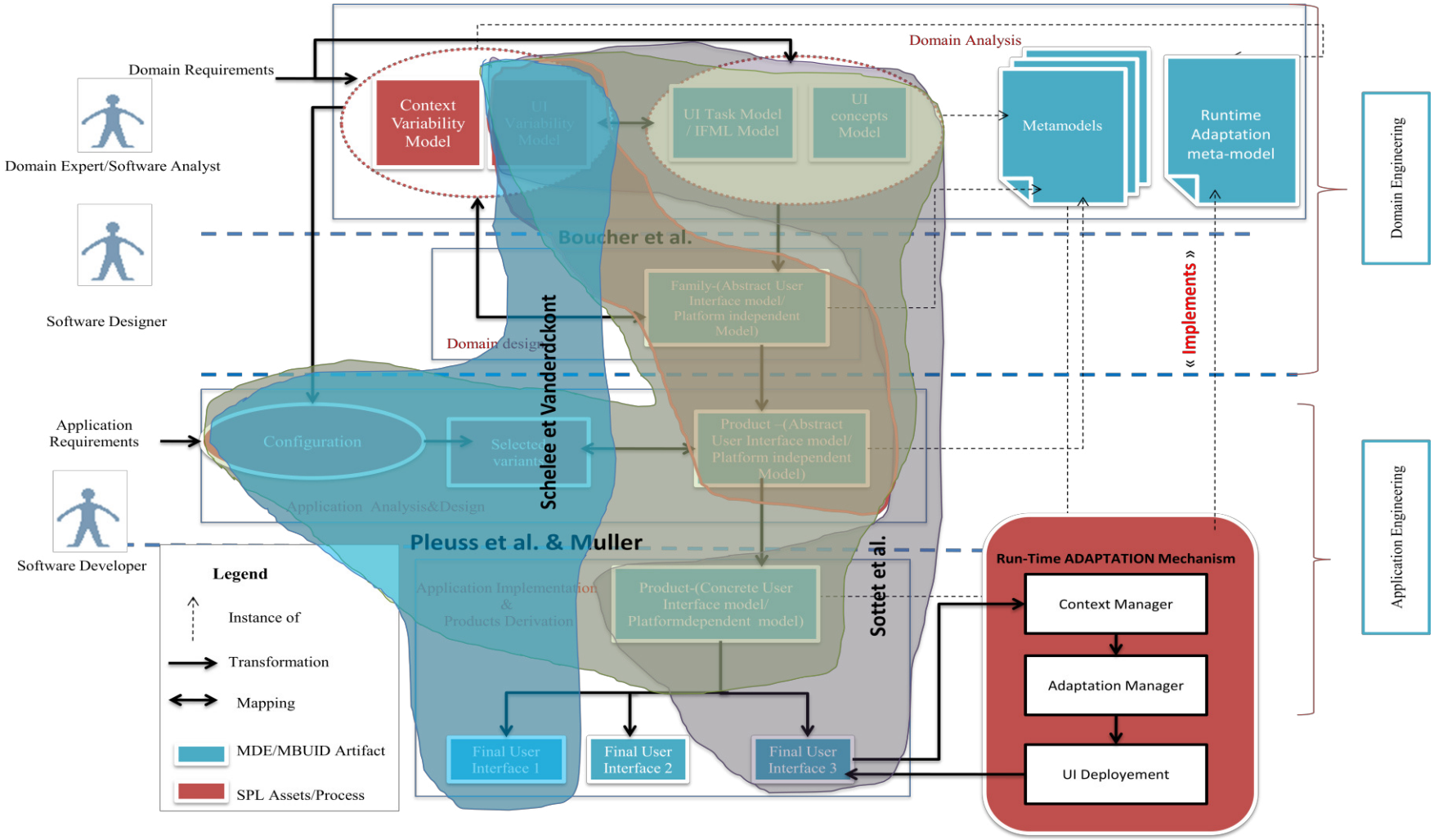


Figure 2-2 The tagging of existing works-Part1

For the second positioning and in Figure 2-3, Logre et al. [42] use a meta-model to define concepts of dashboards, from this metamodel, they generate the feature model that depicts the technological variants of dashboards. Then, from the FM, dashboards are generated.

For Garcés et al. [28], they use metamodels to define the business and the architecture aspects of a GUI. They, also, define feature models to describe the variability of the interface.

The code is generated from the java model. This later is resulted from the mapping between the architecture feature model and the architecture model.

In Quinton et al. [58], authors use the feature model to manage the functional and the technological variabilities, they use models to define features assets and a meta-model to manage the gap between the functional variability and the technological variability.

In the third positioning and in figure 2-4, Arboleda et al. [6] uses MDE core assets (metamodel, models and transformation). Metamodels to define concepts of problem and solution space, models to implement the application level and transformations to generate the final product. The variability model is also used to manage the variability of the system and to derivate the final product.

Gabillon [26] and Kramer [40][41] use feature models to describe the context and the UI variability. As implementation technology, Kramer uses documents while Gabillon use components (both technologies are absent in the summary schema). Furthermore and in order to support feature attributes in the featureIDE platform, Kramer [40][41] extends the feature metamodel.

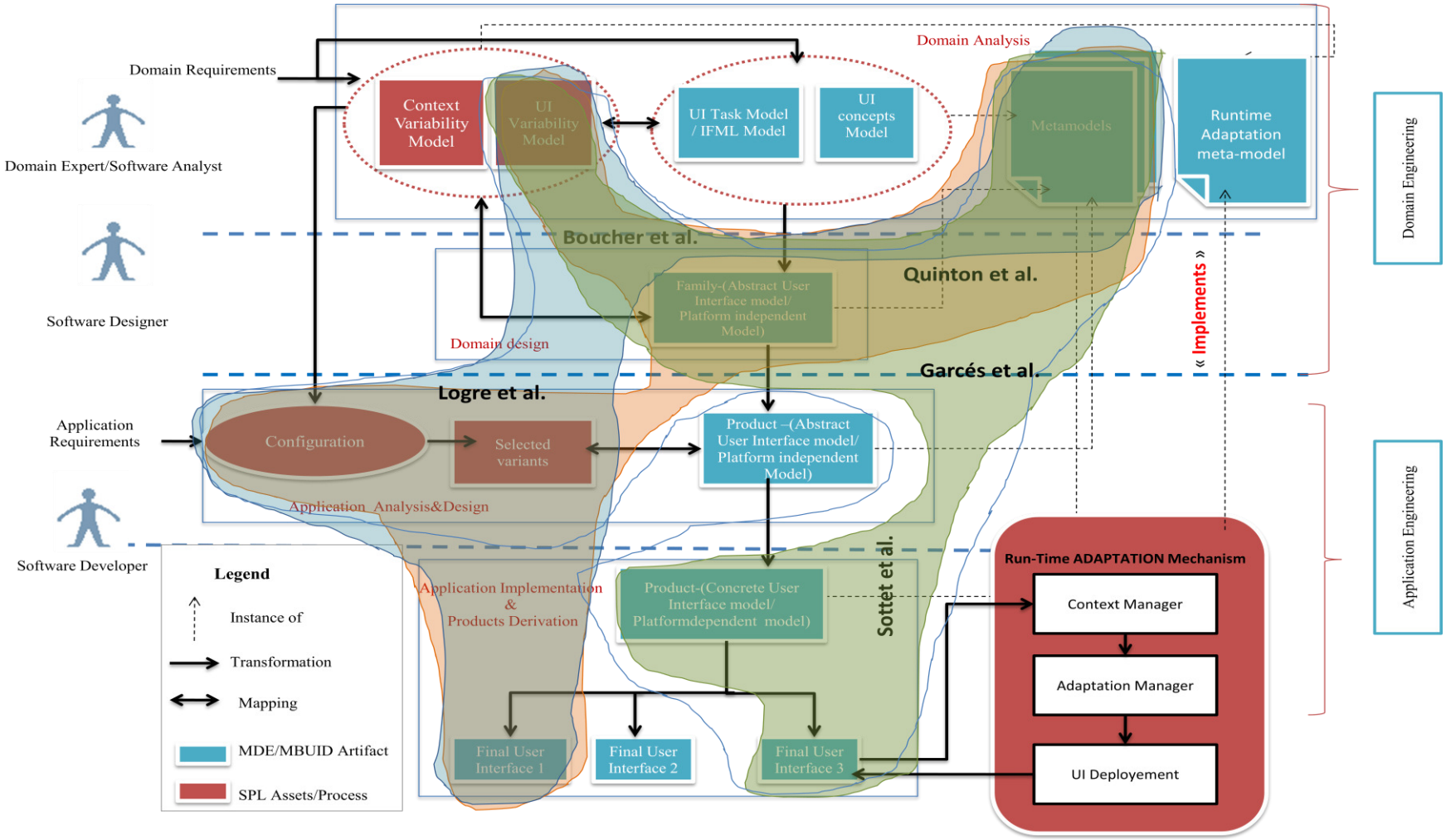


Figure 2-3 The tagging of existing works-Part 2

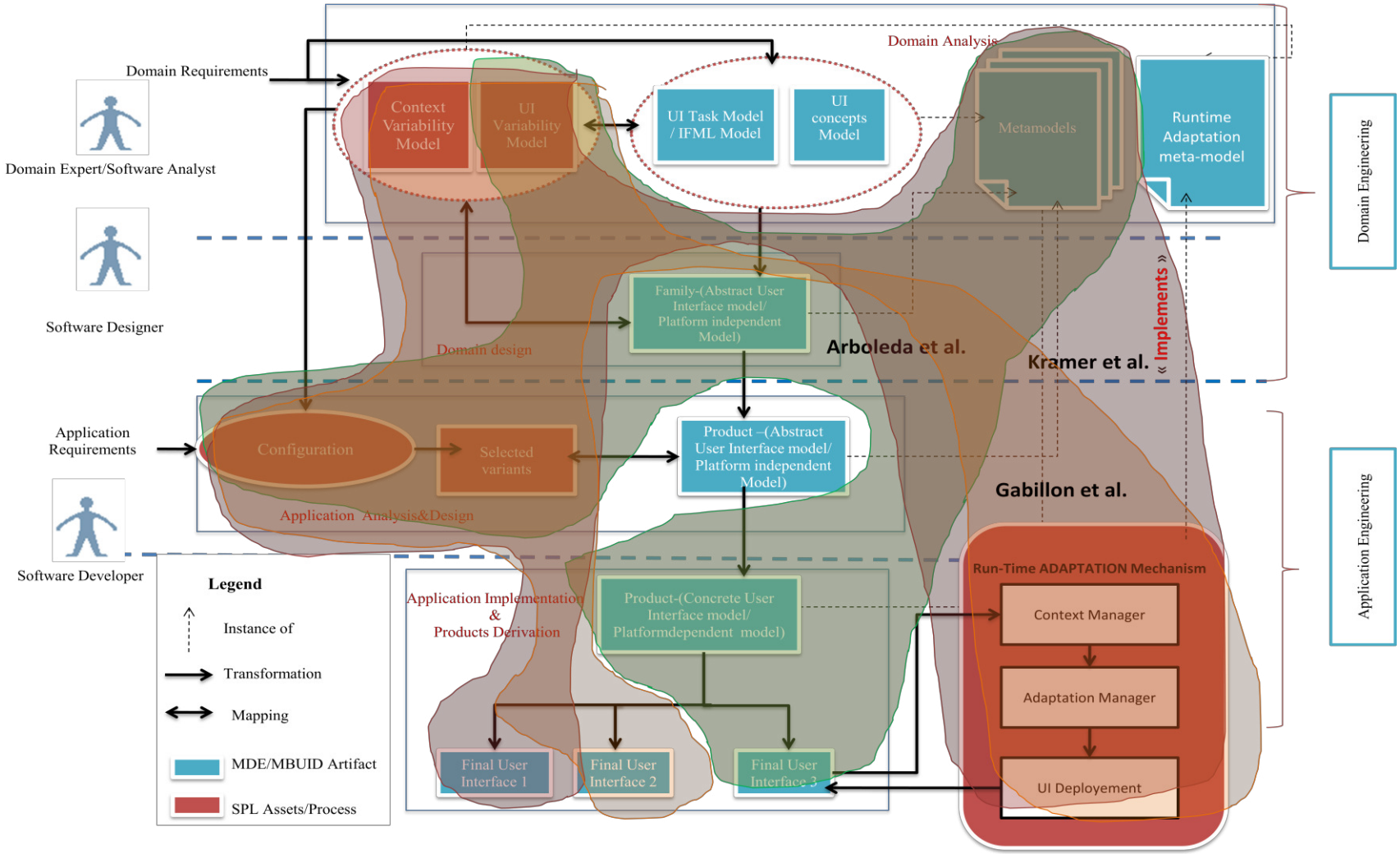


Figure 2-4 The tagging of existing works-Part 3

Based on the previous analysis of UI-SPL approaches and on the positioning schemas, we note that very little works had considered the context of use and managed UI adaptation [26][40][41] [53][55]. In the next section, we focus, in particular, on context-awarness/adaptation of user interfaces.

2.5 Context awareness and UI adaptation in UI-SPL approaches

In this section, we focus on the context consideration within UI-SPL approaches. The approaches compared in table 2-3 use a list of criterias presented as follow:

- **Time of context consideration**: specifies if the context was considered at the design phase, at the runtime phase or at both phases;
- **Context type**: specifies the type of the managed context (i.e. sensed, derived, or profiled);
- **Context element**: specifies the targeted context element (i.e. user, platform, environment);
- **Context consideration/ adaptation techniques**: specifies the technique used to design context and implement the adaptation mechanism.

2.5.1 Context consideration at the design phase

All approaches of table 2-3 have considered the context at the design time. However this consideration differs from one approach to another according to the purpose of context consideration and according to the technique used to design the context.

In [53][55], the context awareness was performed at the design time phase in order to generate a customized UIs. The standard context triplet was extended with a fourth element which is the

customer element. In addition to the end user, there is also the customer who owns a concrete product. Often, this is not the end user itself. The UI customization was performed during the development process by addressing different UI aspects such as the layout aspect, the navigation aspect, presentation unit aspect, UI elements aspect and so one. The UI customization was mainly designed using additional context models. For instance, we find the navigation model, the clustering model, and the arrangement model. Others aspects (such as the abstract user interface elements) are customized during the derivation of a specific UI or at within (the concrete user interface elements) the transformation connecting the AUI model and the CUI model.

Table 2-3 An Overview of context-aware UI-SPL approaches

Approach	Time of Context consideration	Context type	Context element	Context consideration/adaptation techniques
[Pleuss et al. 2013] **[Pleuss et al. 2012]**	Design time	None	<user, platform, environment, **customer**>	MDE models
[Kramer 2014]	Mixte time	Sensed	<user, **platform**, environment>	Design time: context feature model separately designed/ UI configuration Runtime: features/ feature-based corse-grained compositional technique
[Gabillon et al. 2015]	Mixte time	Sensed	<user, **platform**, environment>	Design time: Context and UI features combined under the system FM/ UI configuration Runtime: feature/ a component-based compositional technique

In [40][41] and [26], the design time context consideration was performed in order to adapt the UI to a default context of use. The UI is configured according to the target device on which the application runs. In both proposals, the context was presented using feature models. In [40][41],

the context feature model was separately defined from the UI feature model. While in [26], context feature and UI features was combined under the system feature model.

2.5.2 Context consideration at the runtime phase

Only two approaches [26][40][41] have addressed the context consideration at the runtime phase. The context was handled in order to adapt the UI to the context changes. The sensed data are relative to the platform element. For example, in [40][41], sensed data were about the battery, the connectivity, the telephony, internet and the data synchronization. While In [26], sensed data were about the screen size of the device. In both approaches, the context was presented using features. To recompose the adapted UI, both approaches have used the same technique (compositional technique) but different technologies (In [40][41], Kramer opted for the reuse of feature-based coarse-grained modules called Dynamic Binding Units (DBUs) [59], while in [26], authors have used component).

2.6 Conclusion

As conclusion and based on tables 2-2 and 2-3, and on the positioning schemas, we may list the following shortcomings:

1) The context consideration within UI-SPL approaches is hardly covered, only 4 approaches from 11 approaches deal with context-awareness in UI-SPL approaches. Furthermore and at the runtime phase, the context was managed only in two approaches [26][40][41];
2) To implement the SPL process, the existing approaches have used different technologies such as aspects, components, documents, and models. The use of models rather than any other implementation technology makes the SPL process more abstract and more reusable.

However, the use of models in SPL approaches, in particular specific UI models (MBUID models) is still not widely used.

3) In context aware proposals, the targeted context element was the platform element in [26][40][41] and the customer element in [53][55]. There is no proposal which target the user element (for example his preferences).

For all these reasons and in order to fulfill the listed gaps, we will present in the next chapter the contribution which best meets these lacks.

Chapter 3. A UI-DSPL Approach for the Development of Context-Adaptable UIs (The Design Phase)

3.1 Introduction

In this chapter and based on previously presented works, we present an overview of the proposed approach and its main contributions. Our approach includes two phases: the design phase and the runtime phase, these phases fulfill the gaps of existing works. In this chapter we put the focus on the contributions of the design phase. This phase is reserved for the generation of initial user interfaces. Furthermore and to better describe the design phase, we present a case study which will illustrate this chapter and the next dissertation's chapters.

3.2 An overview of the whole UI-DSPL approach

To fulfill the gaps of existing works, the core aim of this thesis was to bring a DSPL approach to develop context-adaptable UIs [62]. Our approach includes two phases, a design phase to generate the initial user interfaces and a runtime phase that adapts the generated UI to context change. Before presenting the phases of our contribution, a number of contributions were achieved, including:

1) **Contribution 1: Design and implement a profiled context.** Our proposal will deal with profiled context. At the design phase and contrary to sensed context, context features will be implemented (i.e. associate implementation artefacts to context features) in order to generate the context interface. This interface will serve to manually acquire context information at the runtime phase.
2) **Contribution 2: The managed context element is user preferences**. These preferences will be defined vis-à-vis the presentation aspect and the functional aspect of a user interface. At the design phase, user preferences will be designed using the context feature model and at the runtime, they will be managed using a setting interface.
3) **Contribution 3: Make the UI-DSPL approach more abstract**. For that, we will combine MBUID concepts with SPL concepts. Those models are used at the design phase as artifacts to implement the UI and at the runtime to recompose the adapted UI.
4) **Contribution 4: A design pattern for the runtime adaptation mechanism:** To best support the runtime adaptation of UIs. We propose a model which describes the main concepts presenting the runtime adaptation in UI-DSPL approach. The model can be used by developers/designers as a design pattern which automates the implementation of the runtime adaptation mechanism and facilitates its maintenance.

In the following, we present the design phase and its contributions. The runtime phase will be presented in the next chapter.

3.3 An Overview of the design phase

As depicted in figure 3-1, the design phase is presented according to domain engineering and application engineering processes used in a typical SPL process. The domain engineering refers to the design and development of user interface variability. The application engineering refers to UIs derivation and the reuse of 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. At this phase, the variability is defined using the Feature Oriented Domain Analysis (FODA) notation. Two features models are defined, the context feature model and the UI feature model.

For the domain implementation phase, it is made up of reusable artefacts and their correspondent code source implementations. At this phase, the feature models (described during the analysis phase) are implemented using the Concrete User Interface (CUI) [13] 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 and in order to derivate specific UIs, the feature models are configured. Then, we use a model composer that merges artefacts corresponding to selected features. The composed UIs are then transformed, via code generation techniques, into a Final User Interface (FUI) model. A FUI is a representation of the UI in any programming language or mark-up language ready for compilation or interpretation.

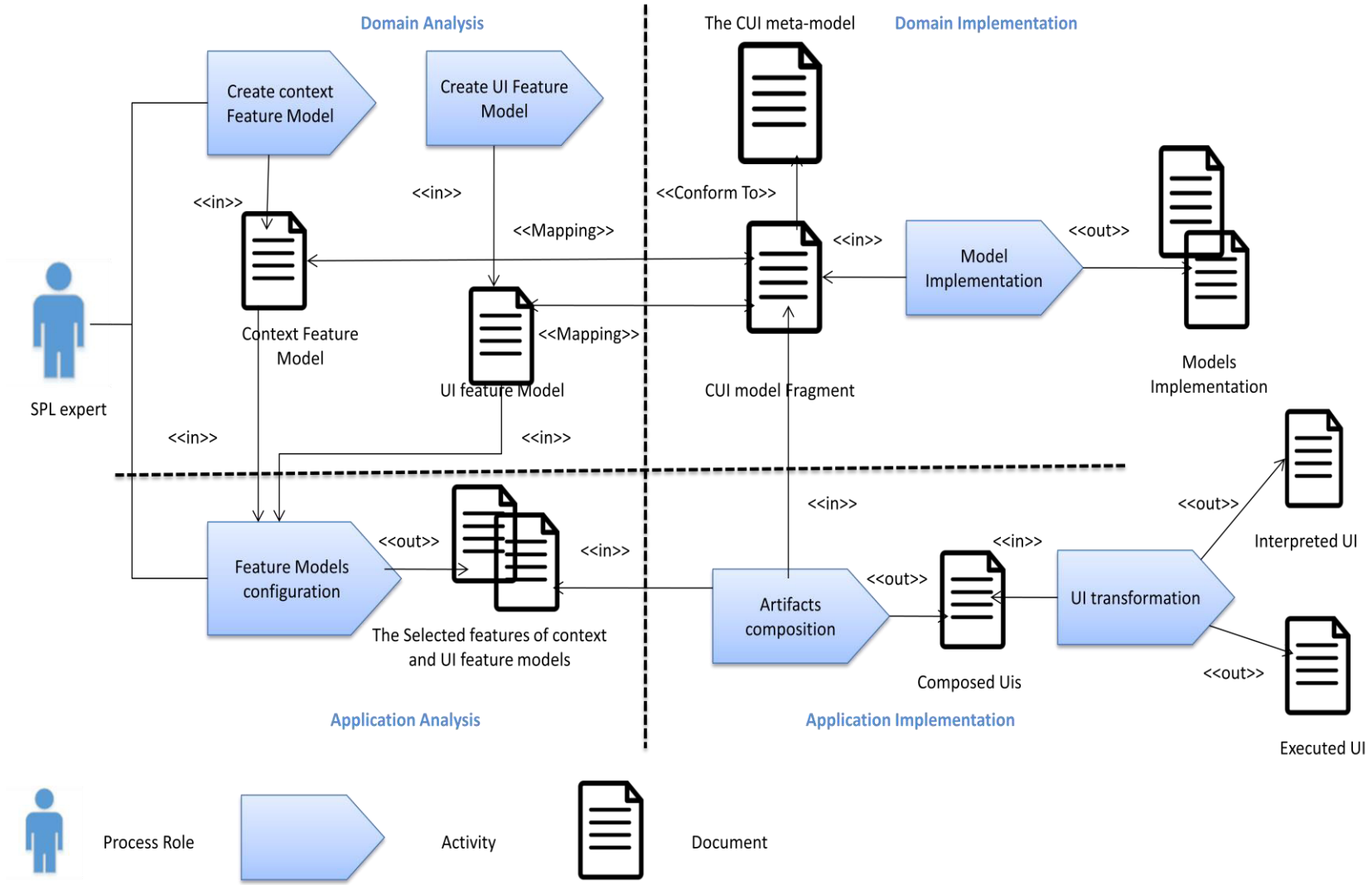


Figure 3-1 The design phase of the UI-DSPL approach, SPEM [69] presentation

3.4 The illustrative example

To better detail the design phase and the runtime phase (see next chapter) of our approach, we present in this section an illustrative example. This example highlights the adaptation of the main interface of the "search for restaurant" application to user preferences change. User preferences are a contextual data specific to application's 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. UI elements, presentation unit), the layout (disposition of UI element on the container), color, sizing, and so one;
- The behavioral aspect: customization of UI element by injecting alternative JavaScript handler.

The illustrative example is about a "search for restaurant" application (figure 3-2 and figure 3-3). The application has two interfaces: a primary interface for search and a second interface for preferences settings.

The search UI (figure 3-2) includes a text field to enter the restaurant speciality, 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.

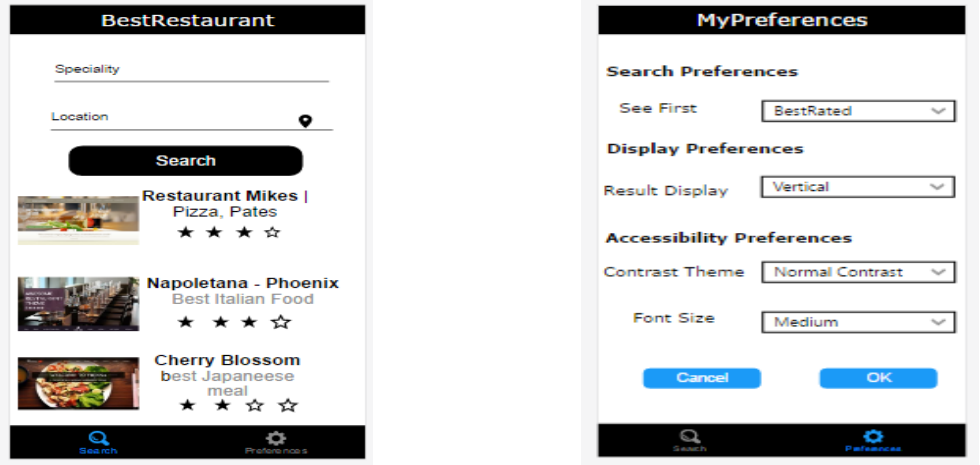
The preference UI (figure 3-3) includes three parts. The first part defines the preferences about the restaurant the user is looking for, the second part defines user preferences about displaying the search result and the third part is about accessibility preferences. "Search preferences"

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 criterias to customize the Theme color and the font size of the search UI.

Default User interfaces are described in figure 3-2 and are relative to the following user preferences:

User1: prefers visualizing the best rated restaurants. The user prefers visualizing the search result displayed vertically in a low contrast theme color and a medium font size.

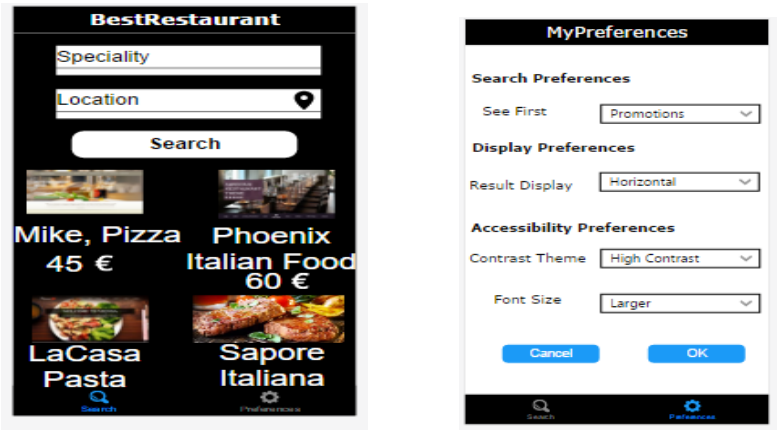


(a) The search UI **(b) the interface of User preference settings**

Figure 3-2 Default UIs of the search for restaurant case study

The first adaptation scenario is described in figure 3-3 and is relative to the following user preferences:

User 2: prefers visualizing restaurant in promotions. The user prefers visualizing the search result displayed horizontally in a high contrast theme color and a larger font size.



(a) The search UI after Adaptation **(b) the new preferences Settings**

Figure 3-3 The search for restaurant UIs after adaptation

3.5 The Design Phase

Two of the most important challenges in Software Product Line Engineering are: variability management and product derivation. The former refers to "how to describe, manage and implement the commonalities and variabilities existing among the members of the same family of software products?". The later deals with "how to build products starting from the selection of a given set of features?". In this section, we describe the domain engineering and the application engineering stages of our UI-DSPL approach from variability model definition until UI generation.

3.5.1 The design phase challenges

1) **Challenge1: Ensure a clear separation of concerns [38][73]**: concerns separation is a fundamental principle of software engineering. Such separation ensures the reuse and the ability to later improve or modify a concern**.** This later can be any part of the system and can be realised as a feature in SPLs. Concern separation will be applied on the feature models (i.e. the context feature model and the UI feature model) of the domain analysis level.
2) **Challenge 2: Use platform independent assets**: as mentioned in the precedent chapter, the concrete user interface (CUI) model is used as an implementation artifact. The use of the MBUID model as implementation artifact will ensure the abstraction and the reuse of the derivation process.
3) **Challenge 3: Specify the composition technique**: to compose feature artifacts, there are two composition techniques. The annotative technique and the compositional technique. To compose our CUI artifacts, we will use a compositional technique based on an XML merger script. This later is used to compose the models (represented in form of XML files) which implements the features of the context feature model and the UI feature model.

3.5.2 The domain analysis phase

3.5.2.1 Concern separation

Concern separation [38][73] is a principle of software engineering. It refers to the delineation and correlation of software elements to achieve order within a system. Through proper separation of concerns, complexity becomes manageable. For example, Object Oriented Programming (OOP)

is a way of separating concerns by decomposing a system into a set of objects that deal with particular functional concerns.

In SPL, concern separation was used by few approaches [40][41], in order to make the system less complex, facilitate its modification, its evolution and ensure its reuse. In SPLs, concern can be realized as a feature.

In the context of our contribution, we used concern separation at the domain analysis level. We design two separated feature models: the context feature model describes the context variability and the UI feature model describes the variability of the main interface (the search interface).

3.5.2.2 Features models and features constraints

Feature modeling is the most common approach to specify product lines. A Feature model is a feature diagram (a special tree of features) plus some constraints. Every node in the tree has one parent except the root feature. A terminal or a concrete feature is a leaf and a non-terminal or compound or abstract feature is an interior node of a feature diagram. Connections between feature and its group of children are classified as “And”, “Or”, and “Alternative groups”. The members of And-groups can be either mandatory, or optional. In the following, we describe the context feature model, the UI feature model and feature constraints relative to the “search for restaurant” example presented above.

a) The context feature model

The context feature model describes the context variability. As shown in figure 3-4, the context variability is expressed across the triplet <user, platform, environment>. “UserPreference” is a sub-feature of the “user” feature and is defined across three features: “SearchPreferences” feature, “DisplayPreferences” feature and “AccessibilityPreferences” feature. The

"SearchPreference" feature defines two variants ("Promotions", and "bestrated" variant), the "DisplayPreference" feature defines two variants ("vertical" and "horizontal" variant).

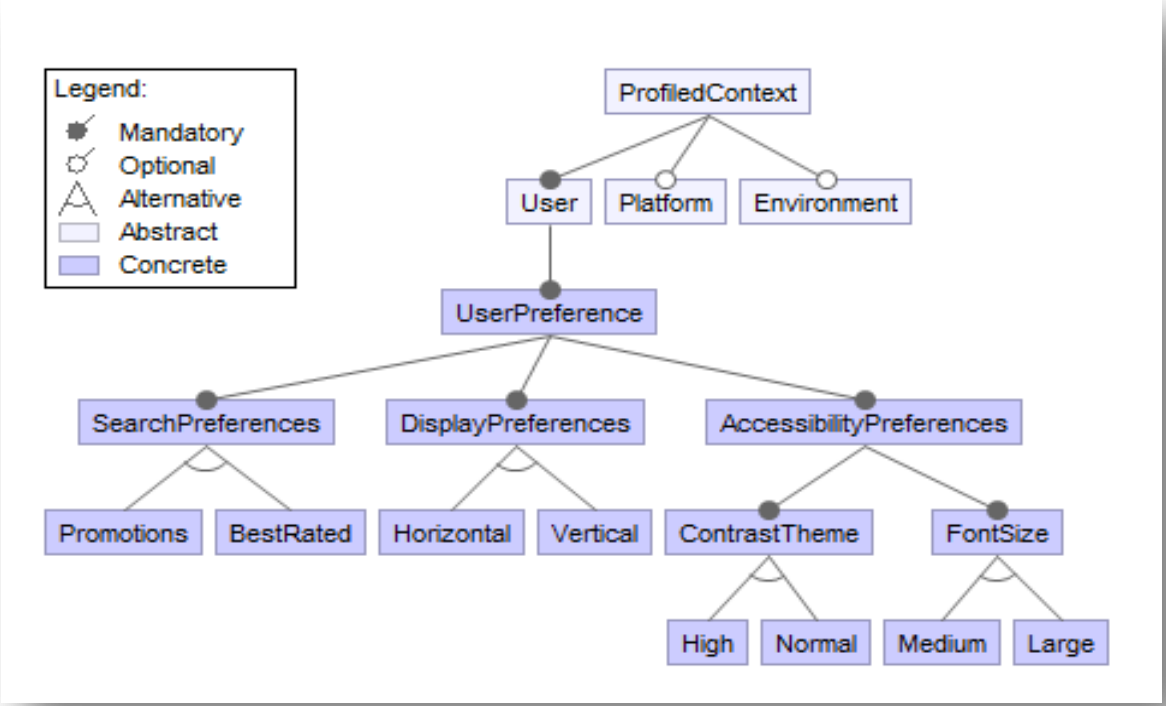


Figure 3-4 The context feature model

b) The UI feature model

UI feature model describes the variability of the user interface [54]. In figure 3-5 the UI variability is expressed across the UI structure and the UI presentation. The "structure" feature defines two variants ("resquestcontainer" variant, and "responsecontainer" variant). The "requestcontainer" has three variant relative to the search widget (the "speciality_textField" variant, the "location_textfield" variant, and the "searchButton" variant). The "searchButton" variant defines the "listener1_BR" variant relative to the listener handling requests of searching the best rated restaurant. Also, the "listener2_P" variant is relative to the "searchbutton" feature

and is relative to the listener handling requests of searching the restaurants offering promotions. The "responsecontainer" feature is defined in terms of widget features and layout features. Widget feature define a"hyperlink" variant presenting the response object of the search request. A hyperlink may be "img" or a "text" link. For the "layout" feature, it presents two variants: the "GridLayout" variant and the "ListLayout" variant.

Regarding the "presentation" feature, this later defines two variants. The "contrasttheme" variant and the "normalcontrast" variant. The former defines a "highcontrast" feature and a "lowcontrast" feature, the later defines a "medium_FS" feature and a "larger_FS".

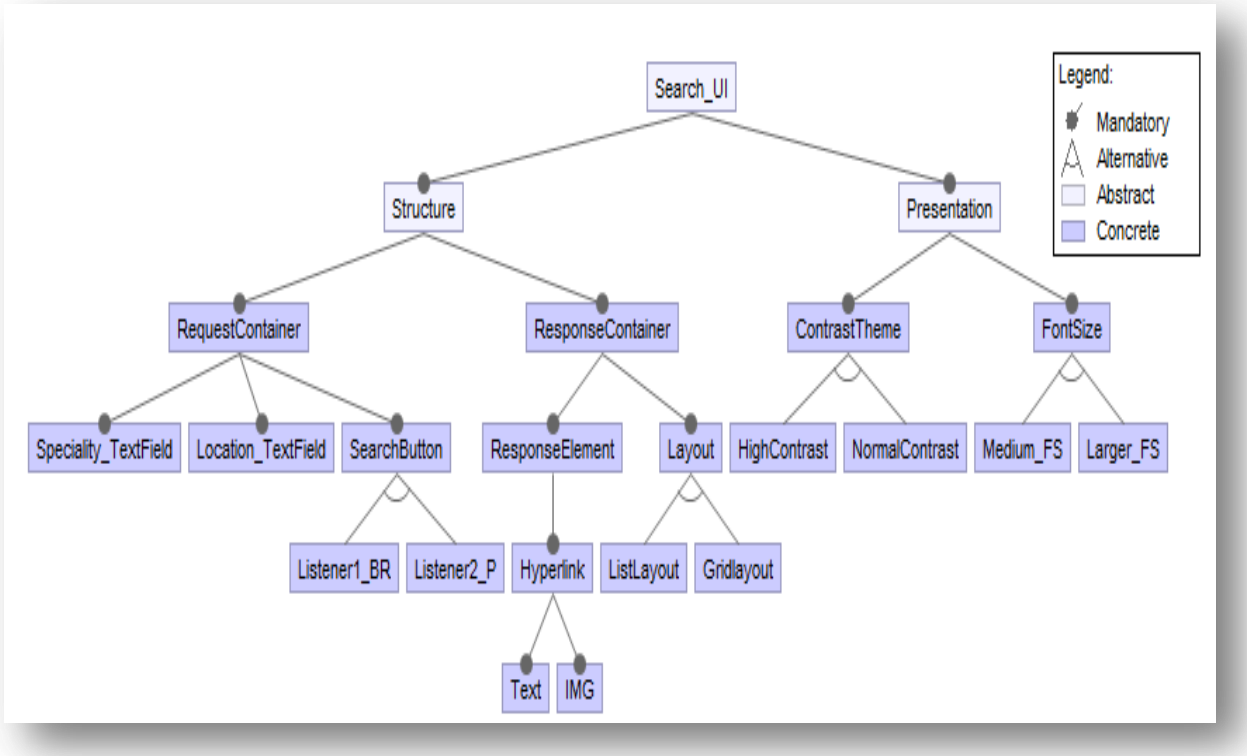


Figure 3-5 The search UI feature model

c) Feature 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 an "intra-feature" constraint) or between features of different feature models (alternatively called a "cross-tree" constraint"). These constraints are used at the design phase to resolve feature conflicts when configuring a feature model and are used at the runtime phase, as adaptation rules, to adapt the interface following the context change. 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 **event** is features presenting the actual context of use, the **condition** is the connection between context features using the "and", "or", or "not" operators and the **action** is the interface features connected using the "and", "or", or "not" operators.

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

Constraint 1: **on** E={promotions, horizontal, high, large} **if** C={ horizontal } **then** A={ Gridlayout }, this constraint 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.

Constraint 2: **on** E={BestRated, vertical, normal, medium} **if** C={BestRated} **then** A={Listener1_BR}, this constraint 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).

Table 3-1 presents the complete list of cross-tree feature constraints describing the link between context features and UI features and used at the runtime phase by the runtime mechanism in order to adapt the running interface. At this phase, cross-tree constraints are, henceforth, called adaptation rules. All these rules are applied following the context change.

Table 3-1 Feature constraints

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

3.5.3 The Domain Implementation Phase

3.5.3.1 Implementation Artifacts (platform independent assets)

The Domain Implementation phase is the process of developing reusable artifacts corresponding to features identified in the domain analysis phase. Implementation artifacts may be a modeling (non code artifacts) or a programming (code source artifact) technology used to implement features in order to develop a family of software products. As example of programming technology, we find feature-oriented programming, aspect-oriented programming, delta-oriented programming, and so one. As example of non-code implementation artifact, we could mention model, and architecture artifacts.

In the context of our thesis and to implement context/UI features, we used the Concrete User Interface model. In the following, we present the CUI metamodel [13] proposed by the UsiXML team. To perform the mapping between features identified at the domain analysis and the CUI model, we propose an excerpt of [13] described in figure 3-7.

3.5.3.2 The CUI metamodel

Figure 3-6 depicts a graphical representation of the UsiXML Meta-model for the Concrete UI. The root entity is *CUIObject* which has been subclassed in *CUIInteractor* and *CUIContainer*. The relationship between Interactors and Containers is captured by the 'contains' relationship and the *CUIRelationship* association class. It is important to note that the meta-model includes specializations for the different modalities (graphical, tactile, vocal), as a CUI Model is modality-dependent. The *Style* class is intended to capture all the presentational attributes for a CUI Object. This design pattern decouples the model from 'presentational vocabularies'.

3.5.3.3 An expert of the CUI meta-model

Figure 3-7 depicts an excerpt of the CUI metamodel of the UsiXML team. In addition to classes described above, we define as “graphicalinteractors”: the “textfield”, “pushbutton”, “imglink”, “textlink”, “radiobox”, “combobox”, and the “label” objects. Regarding “graphicalcontainer”, we mainly defined web containers such as the “footer”, the “header”, the “section”, and the “div” container. Furthermore and in order to manage the user’s “event”, we associate a “listener” class to the “CUIobject” class. The “listener” has to implement the “Action” class in response to the produced event. This CUI model will be used in the next section to implement features of the context and UI feature models.

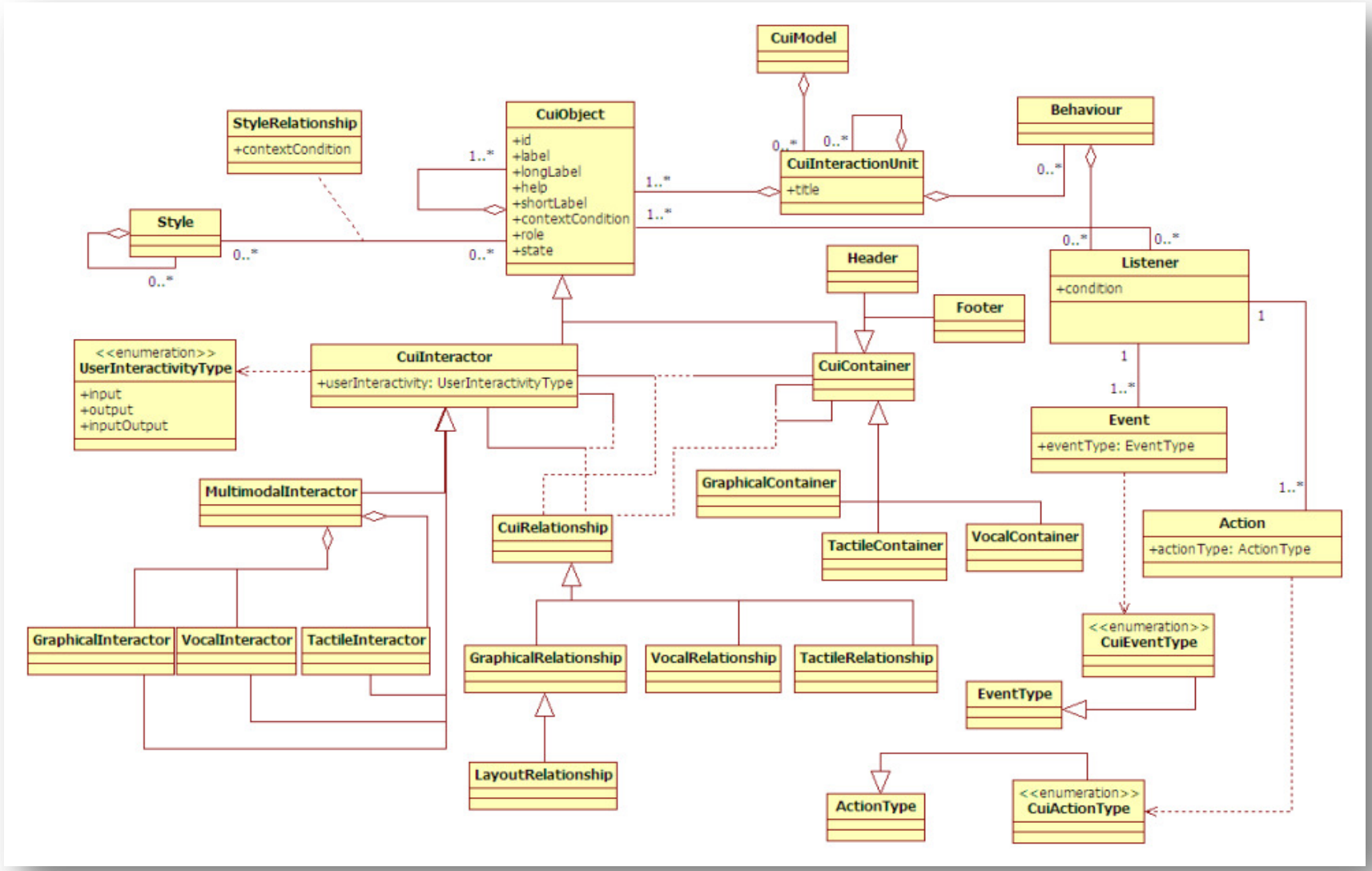


Figure 3-6 The UsiXML CUI Meta-model [13]

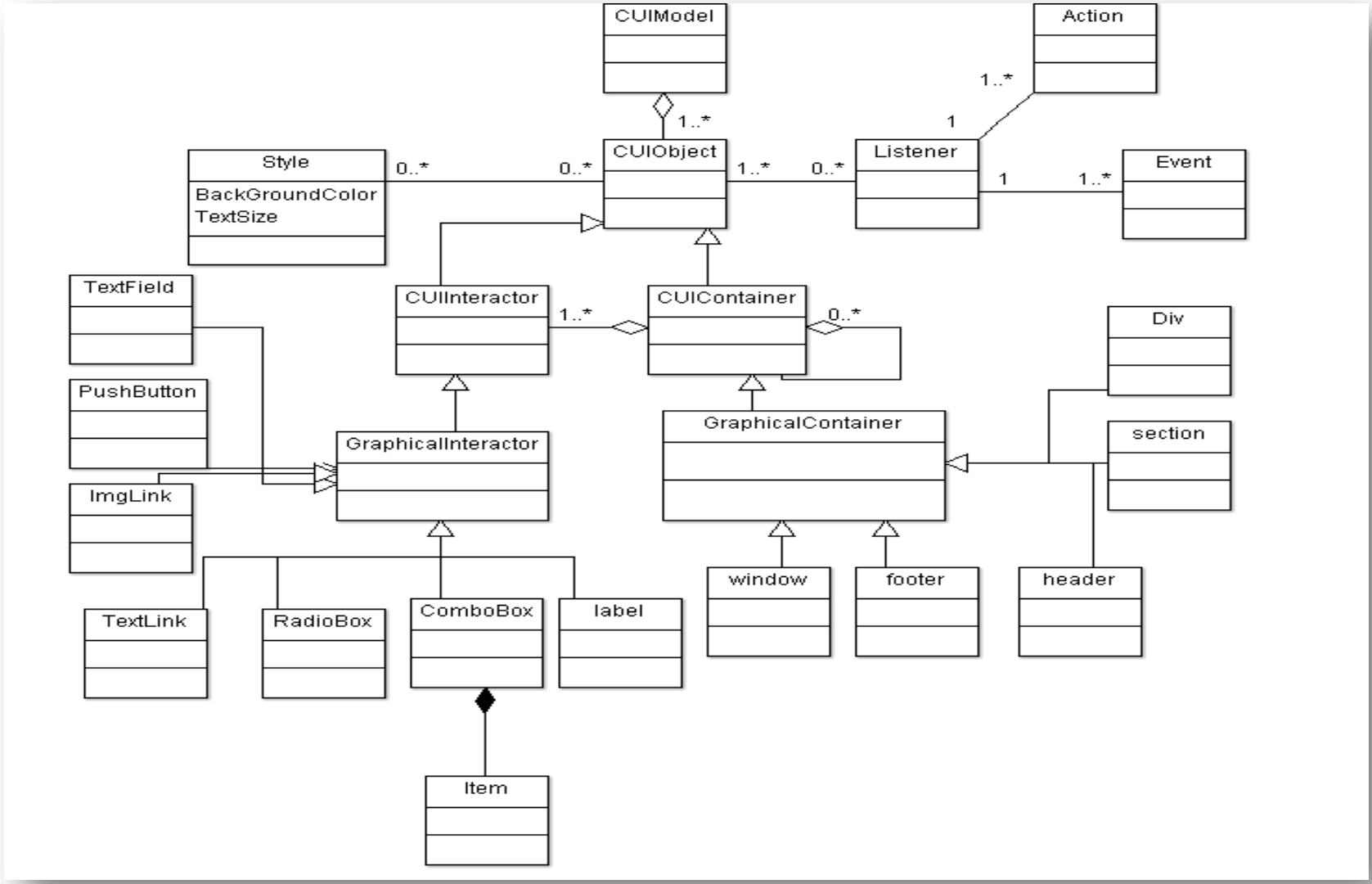


Figure 3-7 An excerpt of the CUI meta-model [7]

3.5.3.4 The mapping between features and implementation artifacts

Figure 3-8 presents the mappings between features of the context feature model and the CUI model. This mapping is applied for "concrete" features, abstract features are not implemented and are used only to structure feature models.

As depicted in figure 3-8, we associate to the "searchpreferences" feature, a fragment of the CUI model. The "searchpreferences" feature is mapped into a model fragment composed by three classes, namely, the main container (the "window1" object), the "div1" container object, and the "label" object, named, the "searchpreference" object. The "searchpreferences" object is contained in the "div1" container and this later in contained, in turn, in the "window1" container object.

Equally, the "promotions" feature is mapped to a CUI model fragment. This later is composed by four objects: "window1", "div1", "CB1", and "promotions". The window1 object contains the "div1" objects which contains, in turn, the "CB1" object. This later contains the "promotions" object.

To perform the mapping feature->model, we take a look at literature. As result, we find very few tools such as the featureMapper [36], the Feature Modeling Plug-in (fmp) [23], the Common Variability Language (CVL) [18]. However, these tools are still in the prototype stage, they do not appear to be working properly, for that we performed the mappings feature->model manually.

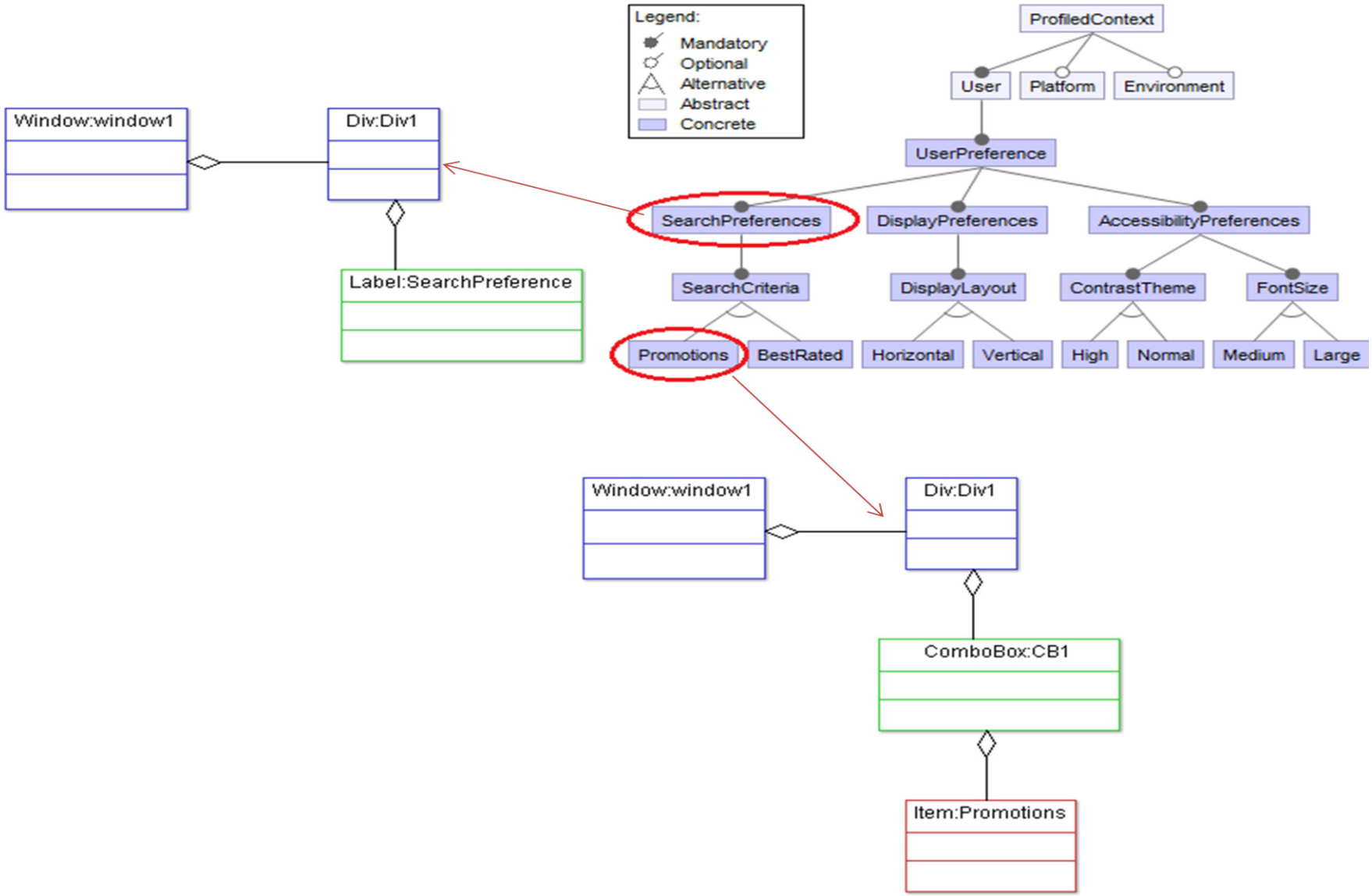


Figure 3-8 Mappings between the context feature model and the CUI model

3.5.3.5 The Final User Interface Model

After realizing mappings between the features and the CUI model, the resulted CUI model fragments are transformed into their source code implementation (the FUI mode). As a target platform, we address a mobile platform. On the mobile platform, the FUI model is described using the HTML markup language. More details about the FUI that presents a specific product are given below. We take note that this transformation is not performed at the domain analysis phase. We allow its realization to the application engineering level.

3.5.4 The application engineering level

This stage is the stage of product derivation. It includes two main phases: the stage of feature models configuration and the stage of product-artifact composition.

3.5.4.1 Feature model configuration

The first step of product derivation is the configuration of feature models. Feature model configuration consists in selecting features that will compose the final product and deselecting others that will not be present.

Figure 3-9 presents the configuration of the context feature model. This configuration is done according to a “default” context of use. This later is defined as following: as “searchpreference”, the user wants to see the “bestrated” restaurants (instead of restaurants in “promotions”). For “displaypreference”, the user wants to see the search result displayed in a “vertical” way (instead of “horizontal” way). For “accessibilitypreferences” and about the “themecolor”, the user prefers seeing his interface in a “Lowcontrast” theme. Regarding the “fontsize”, the user prefers seeing his interface with a “medium” text (instead of “larger” text).

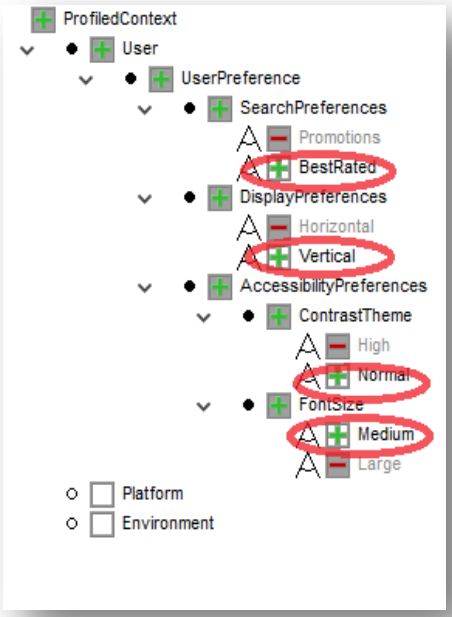


Figure 3-9 The Context model configuration

For the "searchUI" feature model (figure 3-10), it was configured using the default configuration of the context feature model and the appropriate cross-tree feature constraints (table 3-2). According to figure 3-9, if the context feature "bestrated" is selected, this implies the selection of the UI feature, "listener1_BR". This feature presents the listener that will handle the user's request about looking for the "bestrated" restaurants.

For the layout, as the context feature "vertical" was selected, then we have to select the "listlayout" feature. For the presentation aspect of the UI and about the theme color, the "lowcontrast_TC" is selected according to the selected "lowcontrast" feature of the context feature model. While for the UI font size, the "medium_FS" is selected according to the slected "medium" feature of the context feature model.

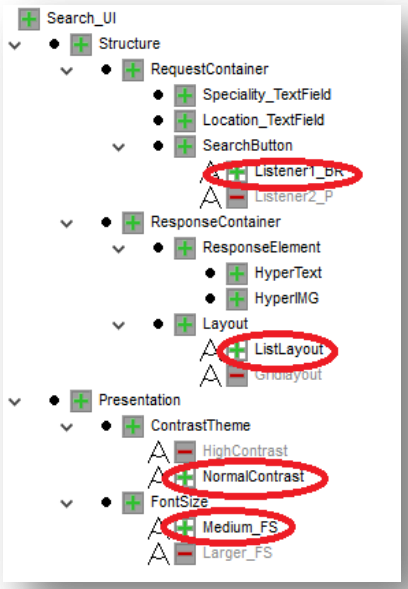


Figure 3-10 The configuration of the search UI feature model

Table 3-2 Applicable feature constraints relative to the default context of use

Rule	Condition	Action
AR1	BestRated	Listener1_BR
AR3	Vertical	ListLayout
AR6	Normal	LowContrast
AR7	Medium	Medium_FS

The feature model configuration is usually carried out using solvers. These solvers serve to resolve conflicts between features models using features constraints (table 3-2) during the feature model configuration. There is different types of solvers such as SAT Solvers, BDD Solvers, Alloy, or SMV. The featureIDE platform uses the SAT solver. Such solver can resolve conflicts between features of the same feature model but not cross-tree features. The conflicts between features of different feature models are not yet resolved. For that, and in our context, the

constraints defined between the context feature model and the UI feature model are resolved manually.

3.5.4.2 Models composition

After the configuration of context feature model and UI feature model, the next step of product derivation is the composition of artifacts implementing the selected features. This Artifact composition amounts to compose the correspondent CUI model fragments. In the following, we will talk about the composition technique.

a) Annotative vs Compositional Techniques

In the field of Software Product Line, there are two composition techniques, namely the annotative technique and the compositional techniques. These approaches provide two different solutions to separating concerns. Annotative approaches focus around the use of virtual separation of concerns [38][73]. Examples of tools that support annotative approaches include the C preprocessor, and CIDE tool. Using these tools, the developer can annotate parts of the source code that are part of each feature in the SPL. This approach can allow very fine grained adaptation including extra statements to methods, and parameter alterations in method declarations. Product derivation is carried out by negative variability. Using negative variability, parts of the system are removed based on which features are present in the product configuration. This causes code to be removed from the final variant of the source code if its associated feature is not included in a product. Compositional approaches on the other hand focus around physical separation of concerns. By physically separating code into multiple modules, these can be composed into different variants at configuration. This approach normally makes use of positive variability because elements that are variable to the product are added to the base product. In this thesis, we opted for the use of the compositional technique.

b) The choice of the composer and UI generation

Since the FeatureIDE platform doesn't include a model composer, we used a separated composition of CUI fragments of model. As depicted in figure 3-11, the idea is to transform the CUI models into their XML representation, then, we will use an XML composer to merge the XML modules which corresponds to selected features. This composition technique is applied for context features in order to derivate the UI context and for UI features in order to derivate the search UI. The resulted XML file is transformed thanks to an XSLT transformation language into an HTML page. More details about UIs composition and HTML derivation are given in the next chapter.

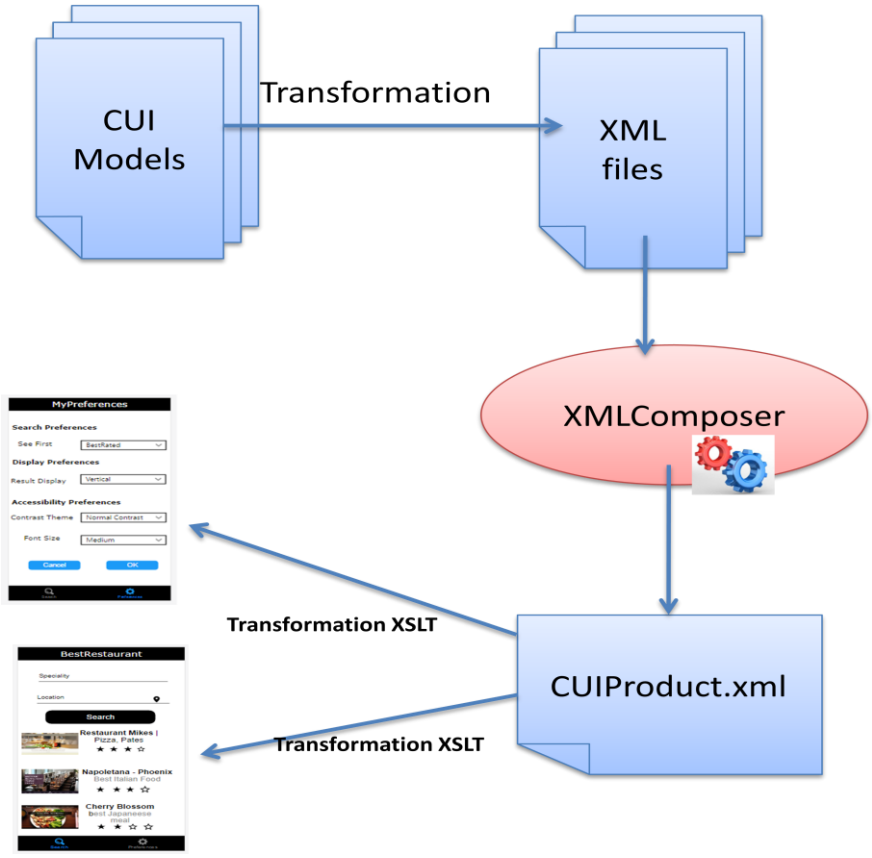


Figure 3-11 Preview of artifacts composition and UIs generation

3.6 Conclusion

In this chapter, we presented first the whole approach, then, we detailed the design phase. The design phase challenges are: concerns separation, the design of platform independent artifacts and the use of a compositional artifact technique in order to generate the final user interfaces. The design phase was detailed according to an illustrative “search for restaurant” case study. In the next chapter, we present the runtime phase of the proposed approach.

Chapter 4. A UI-DSPL Approach for the Development of Context-Adaptable UIs (The Runtime Phase)

4.1 Introduction

In this chapter, we present the second set of our approach's contributions. These contributions are relative to the runtime phase. This later is reserved to adapt the main interface to context of use change. The runtime phase presents two main contributions which are 1) runtime adaptation mechanism and 2) runtime adaptation pattern which will facilitate the design and the generation of the adaptation runtime mechanism.

4.2 An Overview of the Runtime Phase

The runtime phase (figure 4-1) or 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. 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 recomposition;

- The UI code source generator: is responsible for the generation of the UI code source ready for interpretation or compilation.

To facilitate the design and development of the three components of runtime mechanism, An adaptation pattern will be proposed. This pattern describes the main concepts used by runtime adaptation components described above.

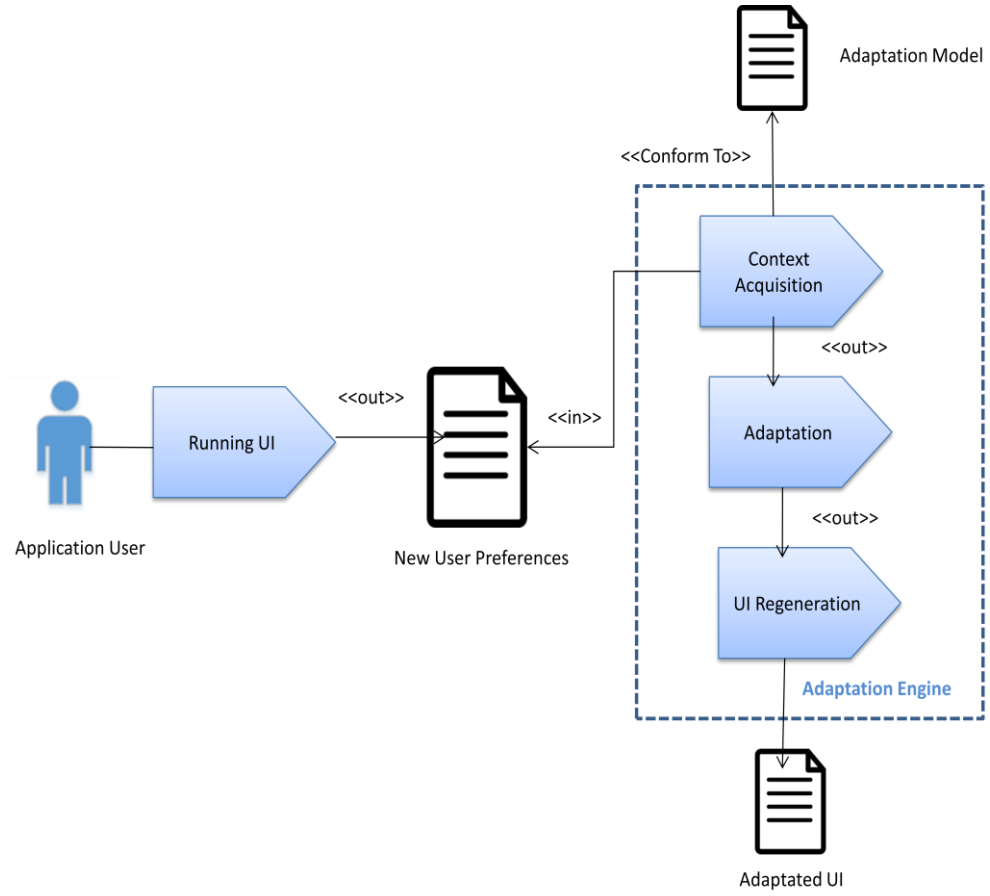


Figure 4-1. The runtime phase of the UI-DSPL approach, SPEM [69] presentation

4.3 The Runtime phase challenges

Unlike a conventional SPL, a DSPL continues to configure and adapt at the runtime. In this thesis, runtime adaptation is caused by context change. Therefore, we identify in this section two main challenges. The first one is the proposition of a design pattern. This pattern is aiming to facilitate the design and the development of the runtime adaptation mechanism. The second challenge is the implementation of the runtime adaptation mechanism aiming to manage the context event, handle the runtime reconfiguration and regenerate the adapted UI.

1) **The runtime adaptation design pattern**: at the runtime phase, we propose a runtime adaptation model in order to facilitate the design and the development of the runtime adaptation mechanism;
2) **The runtime adaptation mechanism**: has to be implemented according to the concepts defined in the design pattern. This mechanism will serve to validate the runtime adaptation design pattern and to adapt the UI in reaction to the changes in the context of use.

4.3.1 The runtime adaptation pattern

In order to facilitate the design and the development of runtime adaptation mechanism, we propose in this chapter an Eclipse Modeling Framework (EMF) model. This model represents a design pattern which can speed up the development of the runtime adaptation process by highlighting the fundamental concepts describing UI adaptation at the runtime and the relation between them. Among these concepts, there are those that represent the core assets already defined at the design phase and that will be reused at the runtime phase. For example, we find:

- UI features: its represent the running configuration. Adapting the UI interface is to find a new configuration for UI feature model which corresponds to context changes;
- Feature artifact: already defined and used at the design phase, the artifacts that correspond to UI features are reused at the runtime phase, according to the new configuration, in order to recompose the new UI;
- Feature constraints: describe the links between features of the same feature model and features of different features models. These constraints will serve as an adaptation rules at the runtime phase.

Figure 4-2 depicts graphically the proposed model. As shown, 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. A transition is performed using adaptation rules.

For states, the model defines three types of UI states:

1) The "Default State" represents the running state. Otherwise, the UI resulted from the design phase;

2) The "Required State" is the UI adapted following context change;

3) The "loading_error" state presents the UI when a loading problem takes place.

A "UIstate" is defined as a set of aspects ("UIaspect") describing the UI at the current time. According to Pleuss et al. [27], a UI aspect 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, etc...).

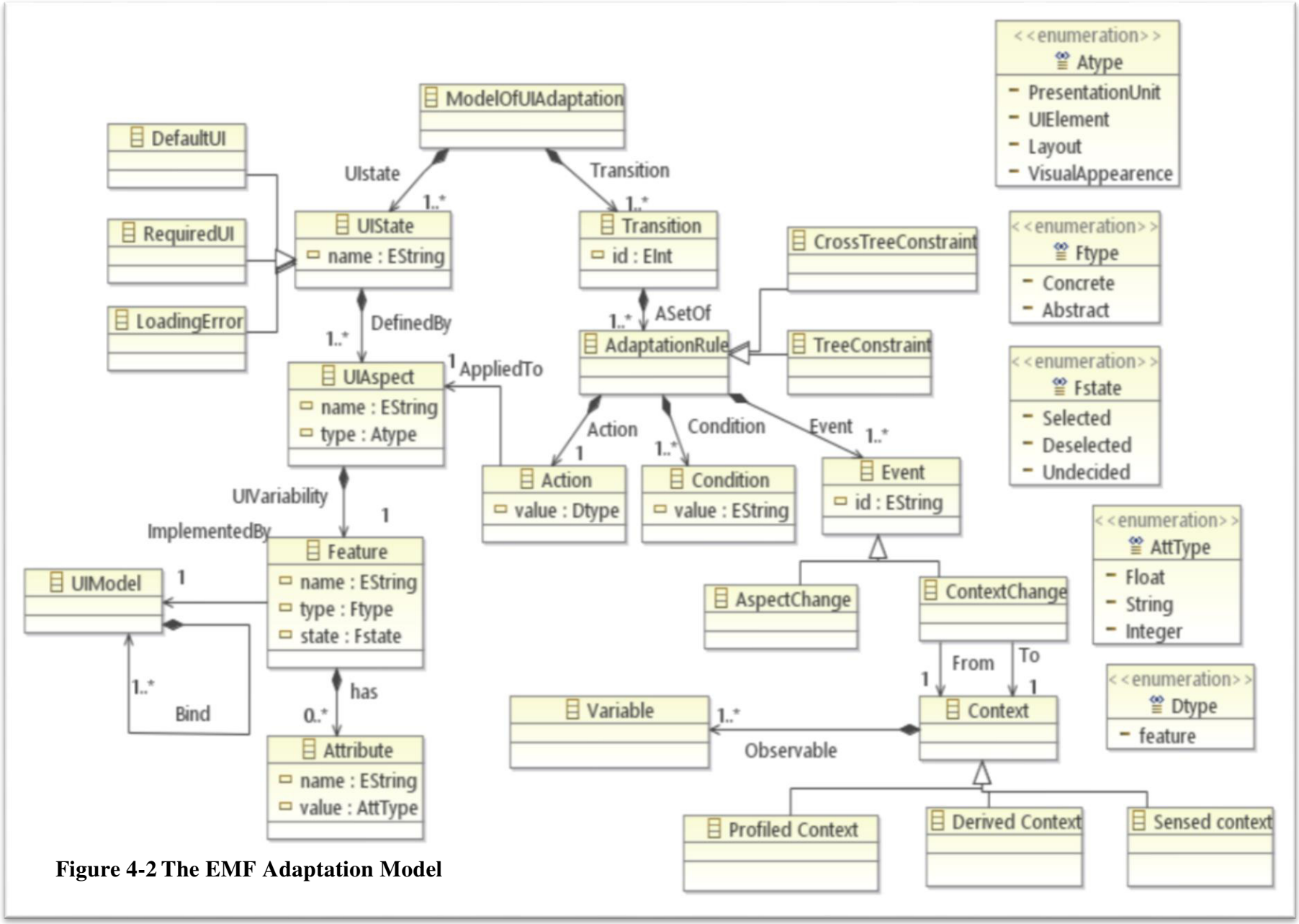


Figure 4-2 The EMF Adaptation Model

At the runtime, these aspects are described using features and their values are described using feature attributes. When an adaptation occurs, another UI state will be defined by updating the aspect features and keeping aspects values.

After UI state updating (another set of features presents the new UI), the UI is recomposed using the "UIModel" artifacts which corresponds to new UI features. The result of UI recomposition is the CUI model of the adapted UI.

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 context constraints (define the link between context features and UI features), complemented by aspect constraints (describing the link between UI aspects and alternatively called constraint propagation rules [20].

Adaptations rules are described as an Event-Condition-Action (ECA) rules:

- An event may be a context change or an aspect change. If the event is a context change, then the rule is a context rule. Else, if the event is an aspect change, then the rule is an aspect rule.
- A condition: following an event, the condition describes the context variable (or the aspect feature),
- Action: if the condition is valid, the action is executed to UI aspects to make the required change.

For enumerative type, the model defines 5 enumerative types:

1) “Atype” specify the type of UI aspects. It may be a presentation unit, a UI element, a visual appearance aspect or a layout;

2) “ftype” specify the type of feature. It may be an abstract or a concrete feature;

3) “fstate” indicates the state of a feature. It may be a selected feature, a deselected feature or an undecided feature;

4) “attType” indicates the type of the attribute value. We used primary data types (float, string, integer);

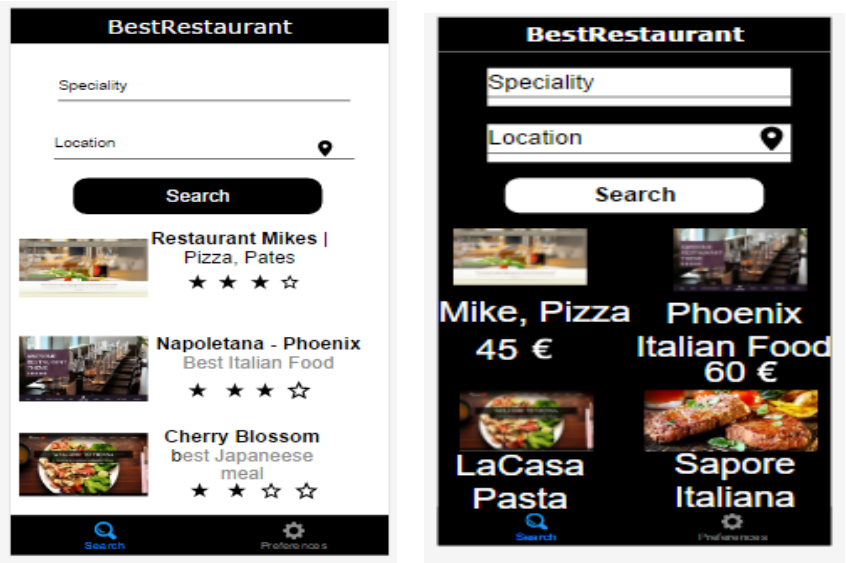
5) “Dtype” indicates the action value, it is a feature.

4.3.2 Model instantiation

To demonstrate the use of the proposed model, we have to instantiate it and implement it (see next chapter). The model instantiation is performed according to the “search for restaurant” use case presented in the precedent chapter. The use case defines two states: a default state conforms to default preferences settings and a required state conforms to new preferences settings. In the following, we give more details about search UI states; adaptation rules allowing the switch from one state to another and context use change.

4.3.2.1 User interface states

Figure 4-3 (a) shows the running search UI (as it was designed at the design phase) while figure 4-4 (b) shows the search UI as it was adapted at the runtime phase.



(a) The Running UI **(b) The Required UI**

Figure 4-3 States of the Search UI

At both phases, the search UI is described in terms of features (figure 4-4, 4-5). At the design phase (figure 4-4), 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. “contrasttheme” feature, “Fontsize” feature);
- Layout (e.g. “layout” feature);

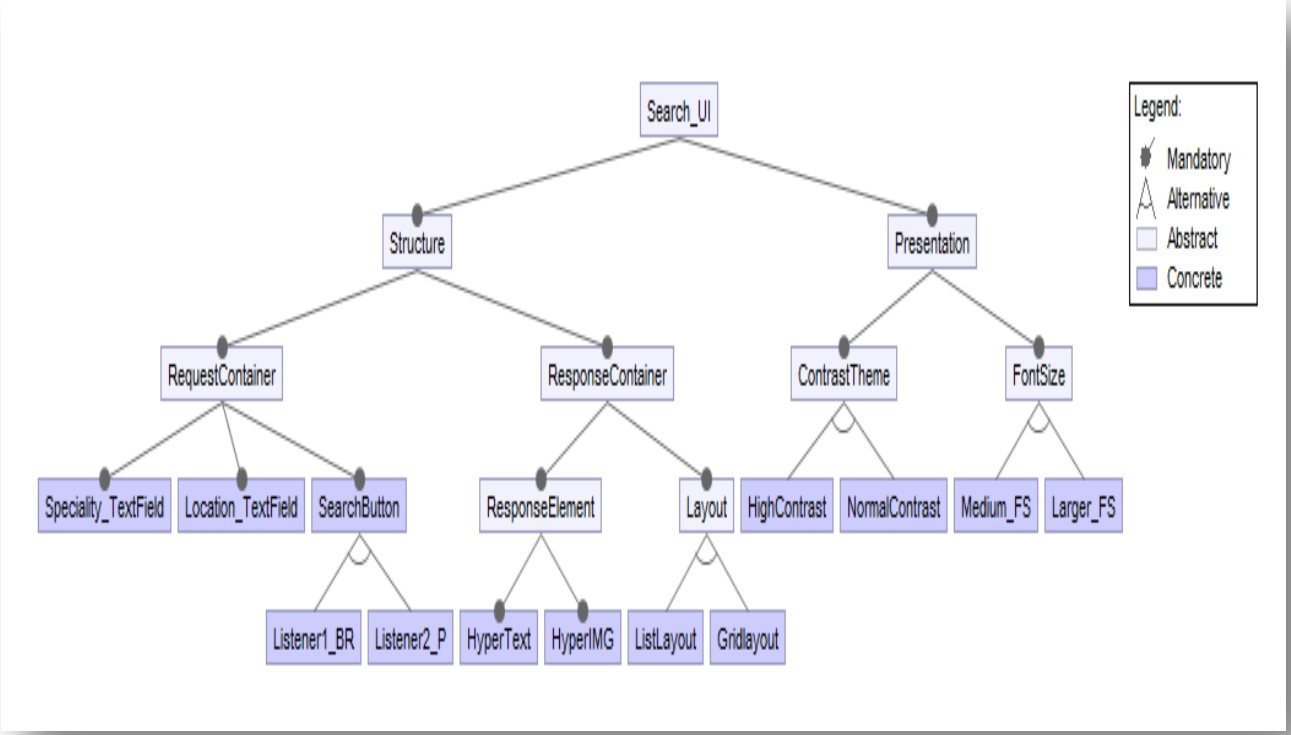


Figure 4-4 The search UI feature model (design phase)

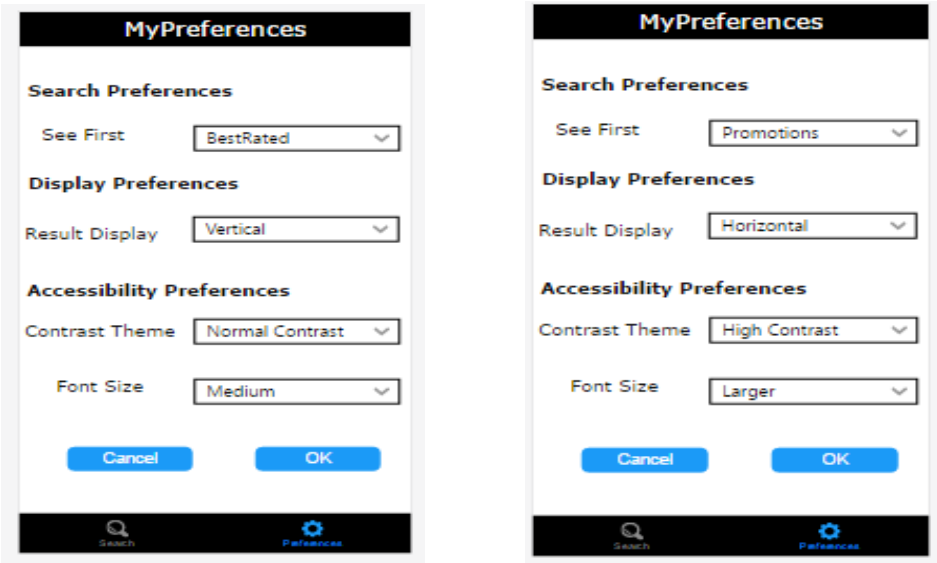
At the runtime phase and in figure 4-5, the "listener_BR" feature is selected to display the best rated restaurant. The "listlayout" feature is selected to display the search result in a vertical way; the "normal contrast" and the "medium" feature are selected to keep the interface with contrast background and to display the text in a medium size.

After runtime adaptation (figure 4-5), 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.

Figure 4-5 Search UI states according to the Adaptation Model

4.3.2.2 The context of use

A UI adaptation is triggered following a context change event. Figure 4-6 shows the change between two contexts of use (the default context and the new context).



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

Figure 4-6 The context of use change

At the design phase, the context of use is presented using a feature model, while at the runtime phase; these features are presented in form of variables.

The context of use as shown in figure 4-6 (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 4-6 (b) shows the user preferences relative 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.

4.3.2.3 Adaptation rules

Table 4-1 shows adaptations rules allowing the transition from the default state of the search UI to the required state. According to the use case, adaptation rules include only context constraints describing the link between context features and the UI aspect features. The condition refers to context value and the action refers to UI aspect feature.

Table 4-1-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

4.3.3 The Runtime adaptation mechanism

Runtime adaptation process was briefly presented at the beginning of this chapter. In what follows, we will give more details about this mechanism by highlighting the managed data and given more details about the components of runtime adaptation engine (figure 4-8).

4.3.3.1 Context acquisition

The first step in the runtime adaptation mechanism is the context acquisition. The context is acquired from the user preferences settings interface (figure 4-7) and is saved as variables in the context data storage (figure 4-8).

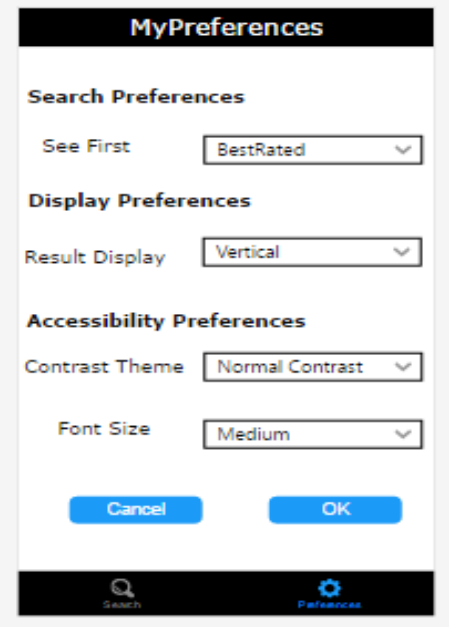


Figure 4-7 The context acquisition interface

4.3.3.2 Reconfiguration Algorithm

After context data acquisition, the adaptation engine looks for a new UI configuration. This configuration is generated using the running configuration and the adaptation rules.

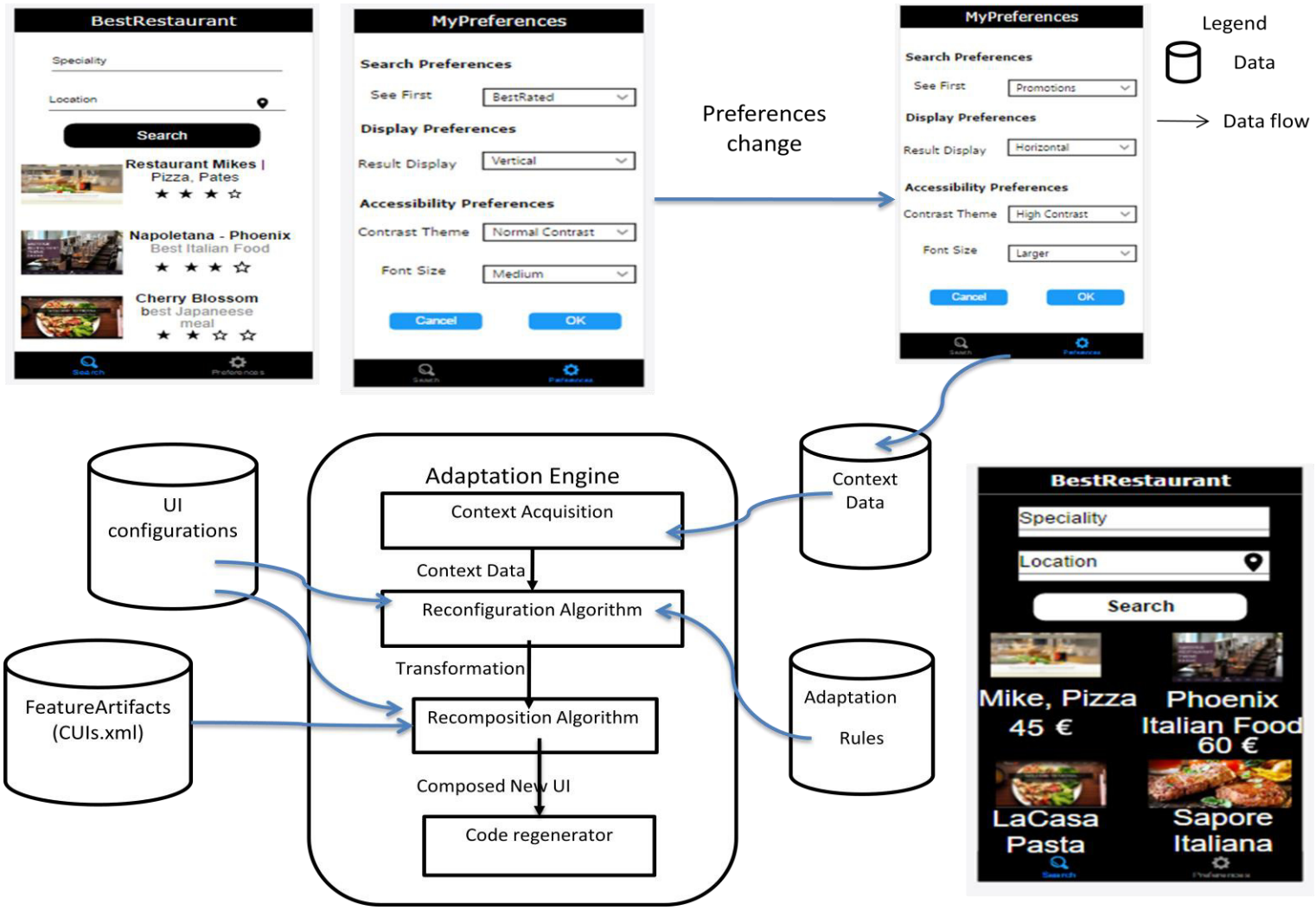


Figure 4-8 The Runtime Adaptation Mechanism

Algorithm 1 illustrates how the new UI configuration is obtained from context information. To start, the algorithm requires a set of updated observable obtained from the context manager, and the current configuration obtained from the product itself. 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 aspect belongs to the current configuration, the algorithm verifies if the new observable value is false. In that case, the aspect has to be unwoven from the product. For the observables whose aspect does not belong to the configuration, the algorithm checks if the new observable value is true. In that case, the aspect has to be woven to the target configuration. After the selection of variants of aspects to weave and the deselection of variants of aspects to unweave, the algorithm obtains the target configuration.

Agorithm1: Runtime Reconfiguration

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

4.3.3.3 Recomposition Algorithm

Once we got the new configuration of the search UI, we have to recompose the new (adapted) UI. Algorithm 2 illustrates how a new UI is recomposed using the artifacts which corresponds to the selected UI features. To start, the algorithm requires the new configuration obtained from the reconfiguration algorithm, the appropriate implementation artifacts resulted from the design phase and obtained from the artifact storage. The algorithm consists in browsing the list of selected feature (saved in the new configuration file). For each feature name, we have to look for its implementation in the artifact's directory. Each time, we find a correspondent (the feature name matches with an artifact file name), we apply a merge. The algorithm iterates over features composing the new configuration file.

Agorithm 2: Runtime Recomposition

Recomposition Algorithm Require: the new configuration
Require: the set of feature artifacts
Ensure: the composed UI Product (CUI.xml file)
1:resultedfile={}
2: for I ∈ features of the new configuration file
3: for J ∈ {list of implementation artifacts}
4: if (value (I)=Name (J)) then
5: merge (resultedfile, content(J))
6: end if
7: end for
8:end for

4.4 Conclusion

In this chapter, we presented the runtime phase of our approach and their main contributions. The runtime challenges are: an adaptation design pattern and a runtime adaptation mechanism. This

phase was detailed according to the illustrative "search for restaurant" case study presented in the above chapter. In the next chapter we will implement both phases of the approach using the appropriate platforms and technologies, and then we will evaluate the generated UIs.

Chapter 5. Implementation, Evaluation and Discussion

5.1 Introduction

In this chapter we present design and development tools used at the design and runtime phases. We also present the realized prototypes. These prototypes were developed for a mobile platform. Finally, we present different methods used to evaluate the generated UIs. We mainly use three evaluation methods: an SPL-based evaluation, a scalability evaluation and a qualitative/quantitative evaluation based on the IBM-CSUQ questionnaire.

5.2 The design phase implementation

5.2.1 Design phase tools

Many tools were developed in order to support feature modeling. Some tools are free and open-source and others are paid tools. As free tools, we find the FAMILIAR DSL (Domain Specific Language) [2], the SPLOT [68] platform, and the featureIDE platform. For paid tools, pure::variant tool [57] is the most known. In our thesis we use the FeatureIDE platform [72] to design and configure feature models.

5.2.1.1 FeatureIDE tool

FeatureIDE is an open-source framework for feature-oriented software development (FOSD) based on Eclipse. All phases of FOSD are supported in FeatureIDE, namely domain analysis, requirements analysis, domain implementation, and software generation. FeatureIDE supports several FOSD implementation techniques such as feature-oriented programming, aspect-oriented programming, delta-oriented programming, and preprocessors. In our context, the FeatureIDE platform was used only for feature modeling (figure 5-1) and for feature model configuration (figure 5-2). Feature mappings/implementation and UIs composition have been performed separately since the featureIDE platform does not support model artifact yet.

a) Edition interface

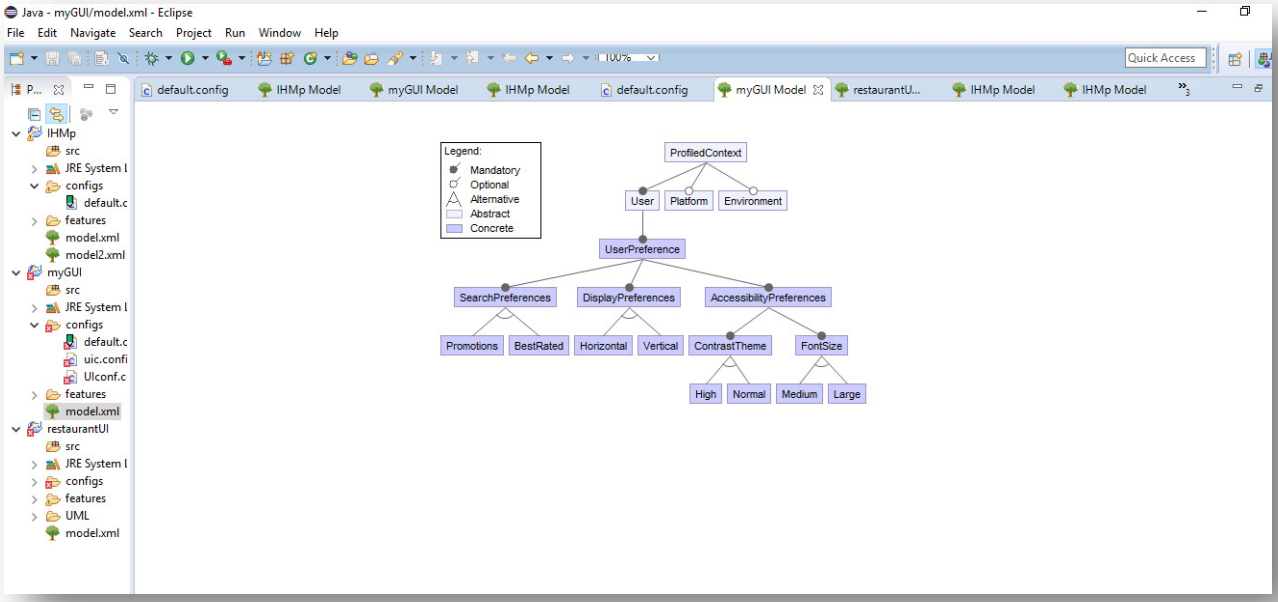


Figure 5-1 The edition interface of feature models, FeatureIDE platform

b) Configuration interface

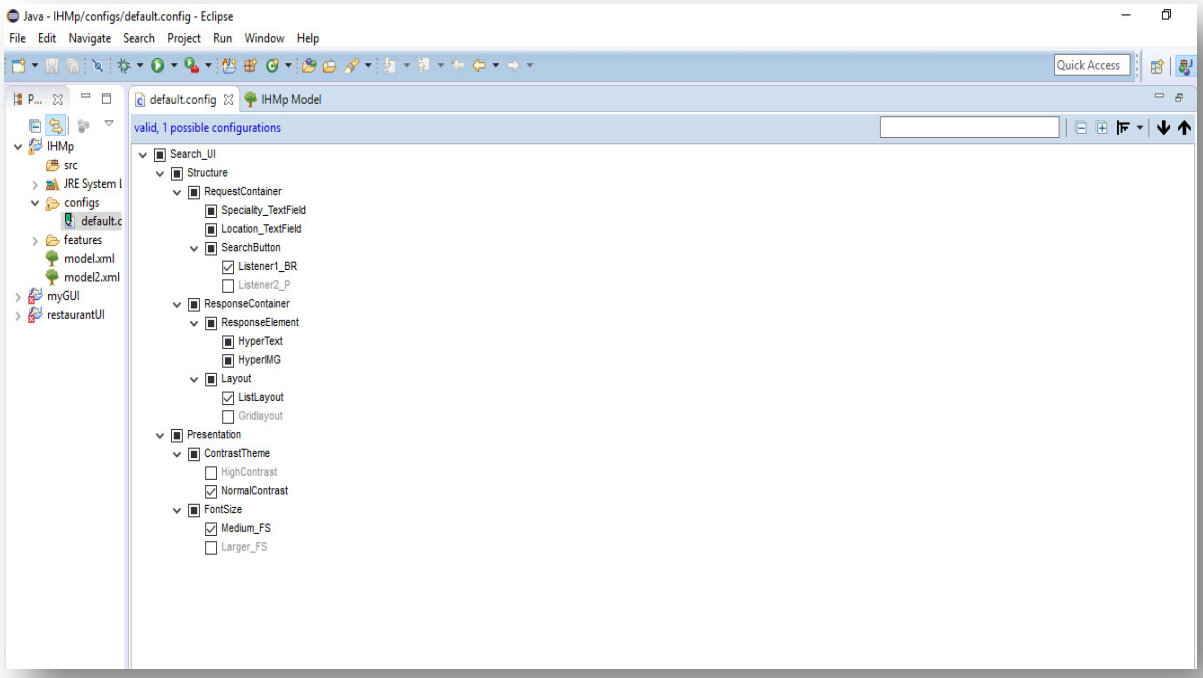


Figure 5-2 The configuration interface, the FeatureIDE platform

5.2.2 GUI models composition

As described in the previous chapter, to compose the feature artifacts, we used our own merge algorithm. This algorithm is inspired from the Olivier Becker merge code and is used to merge the XML representations of the CUI models (features artifacts). In the following, we present the algorithm implementation.

5.2.2.1 The merge algorithm

In his algorithm, Olivier [77] uses the eXtensible Stylesheet Language (XSLT file) to merge two XML files. In our context, and to compose our XML files (the representation of the selected features), we need to install the "saxon" command. In the following, we give more details about "saxon" installation.

5.2.2.2 Saxon installation

Step1: install the Java runtime machine

```
tar zxvf jre-8uversion-linux-x64.tar.gz
```

Step 2: update the system

```
Sudo apt-get update
```

Step 3: After updating the OS run following command to install the package

```
Sudo apt-get install libsaxonb-java
```

Step 4: Execute the saxon command

```
saxon file1.xml merge.xslt with=file2.xml > result.xml
```

5.2.2.3 The merge Script

Before presenting the merge script, we present in figure 5-3 (the feature model of the context) the transformation of feature artifacts into XML code. After this transformation, we use the merge algorithm (Algorithm 1) presented in the above chapter to merge the XML presentation of all feature artifacts. In the following, we give more details about the implementation of the algorithm using Linux Shell. The Shell script has two command parameters: the first parameter is the directory containing XML files and the second parameter is the configuration file which contains selected features. The result of the merge is an XML file (result.xml). Figure 5-4 depicts the result of composition of context features artifacts.

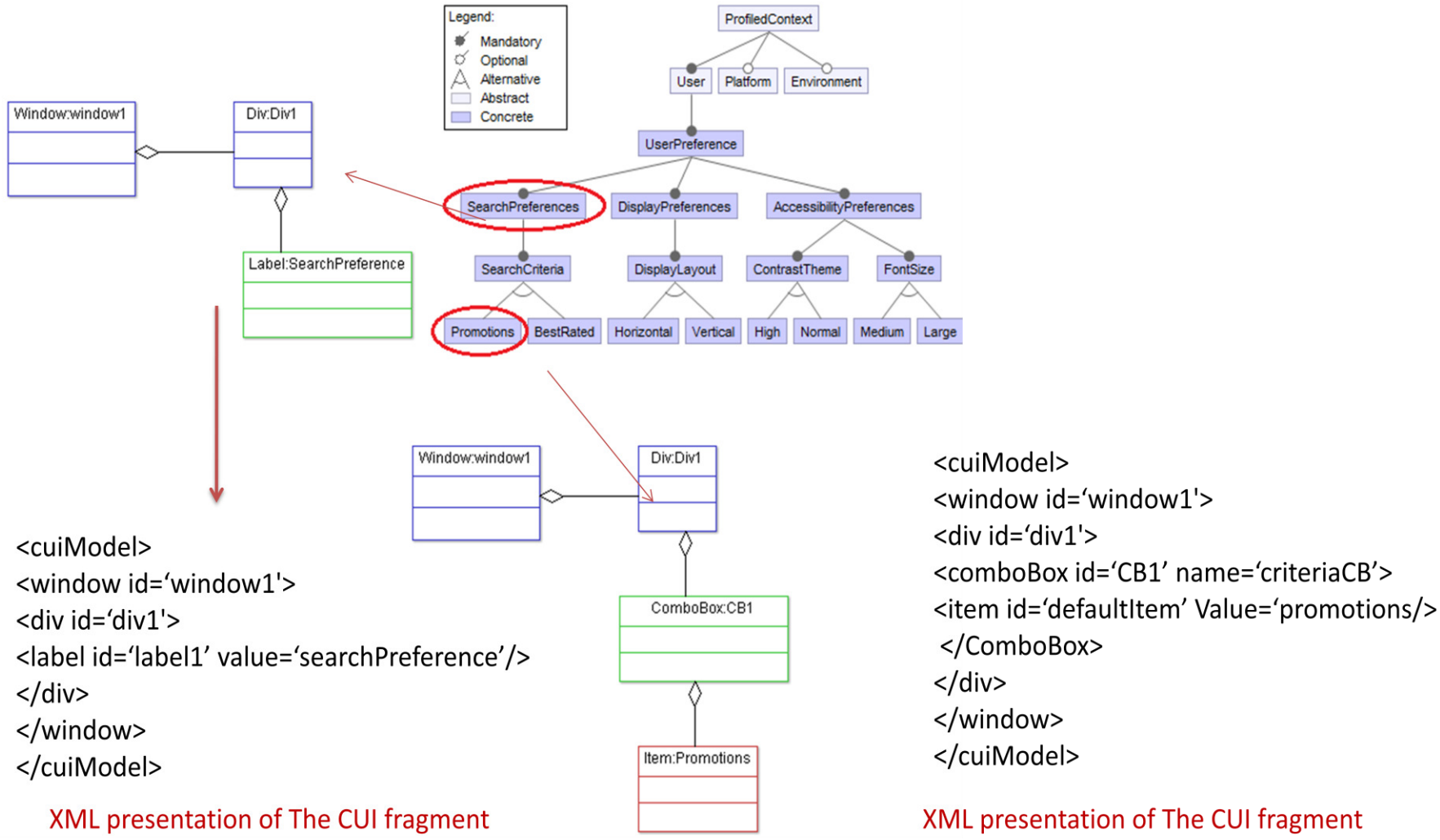


Figure 5-3 The XML representation of the CUI model fragment

```
# THE MERGE SCRIPT


 # !/bin/bash
 #$1 are a command parameter, $1 contains the name of the directory containing
 #the XML files presenting the artifacts of the selected features.
 #$2 contains the name of the configuration file

Touch result.xml # this file contains the code of the composed UI

For I in 'cat $2'
Do

For J in 'ls $1'
Do

If [ $i -eq '$j' ];then

saxon tt merge.xslt with=$i>> aux

cp aux tt

endif

done

done
```

```
1  <cuiModel>
2  <window id='window1'>
3  <div id='div1'>
4   <label id='label1' value='searchPreference'/>
5   <label id='label12' value='seeFirst'/>
6  <combobox id='CB1' name='criteriaCB'>
7   <item id='Item1' value='promotions'/>
8   <item id='item12' value='BestRated'/>
9   </combobox>
10  </div>
11 <div id='div2'>
12  <label id='label2' value='DisplayPreference'/>
13  <label id='label21' value='ResultDisplay'/>
14 <combobox id='CB2' name='DispLayout'>
15  <item id='Item2' value='GridLayout'/>
16  <item id='item21' value='listlayout'/>
17  </combobox>
18  </div>
19 <div id='div3'>
20  <label id='label3' value='accessibilityPreference'/>
21
22  <label id='label31' value='contrastTheme'/>
23 <combobox id='CB3' name='CT'>
24  <item id='item3' value='HighContrast'/>
25  <item id='item31' value='LowContrast'/>
26  </combobox>
27  <label id='label4' value='FontSize'/>
28 <combobox id='CB4' name='FS'>
29  <item id='item4' value='larger'/>
30  <item id='item41' value='medium'/>
31  </combobox>
32  </div>
33 <div id='div4'>
34  <button id='but1'>ok</button>
35  <button id='but2'>cancel</button>
36  </div>
```

eXtensible Markup Language file

Figure 5-4 The XML file of the context of use Interface

5.2.3 CUI-FUI transformation

Since the UI composition is realized, it's time to generate the HTML source code of the composed context user interface. For that, we have to apply an XSLT transformation on the composed XML file (figure 5-4). The XSLT source code used to transform the composed context UI is presented in figure 5-5. After coding the XSLT file, we used the following shell command to generate the final context interface (figure 5-6):

```
1) The command of Installing the xsltproc package
2) xsltproc stylesheet.xsl mapage.xml > context.html
```

```
<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 -->

<xsl:template match="cuimodel/window/button">
<input type="button" class="button" value="Cancel"/>
</xsl:template>
<xsl:template match="cuimodel/window/button">
<input type="button" class="button" value="Ok"/>
</xsl:template-->
</xsl:template>
```

Figure 5-5 The XSLT file relative to the generation of the context of use interface

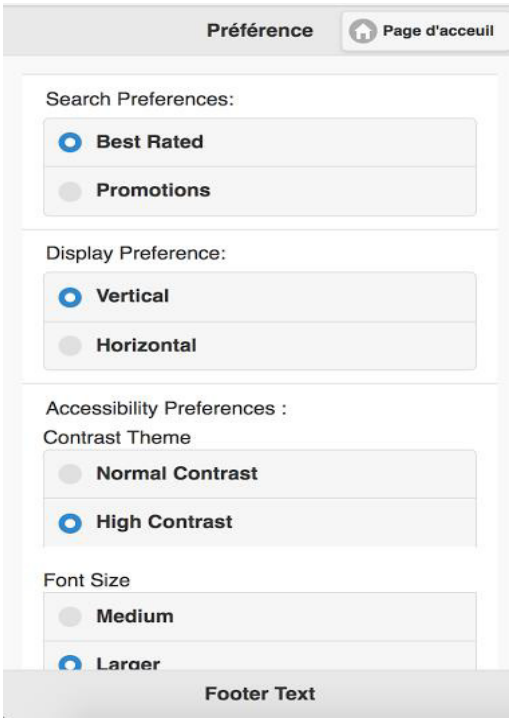


Figure 5-6 Generated UIs (design phase)

For the search UI, more details are given in the appendix A. In this later, you find the XSLT file (which calls two others files: the formatting file "thouraya.css" and the Javascript file "thouraya.jss") and the generated search interface.

5.3 The runtime phase implementation

In this section, we present the implementation of runtime algorithms: the configuration algorithm and the recomposition algorithm. As described below, these algorithms were implemented using the shell script. Their input data (context data, adaptation rules and the running configuration) were managed using files. In addition to algorithms, we present different adaptation scenarios of the main interface (the search interface).

5.3.1 Runtime Configuration datas

The runtime reconfiguration mechanism needs as inputs: the running configuration (figure 5-7), the configuration as it was realized at the design phase, the context data (figure 5-9) and the adaptation rules (figure 5-8). This data is saved in files.

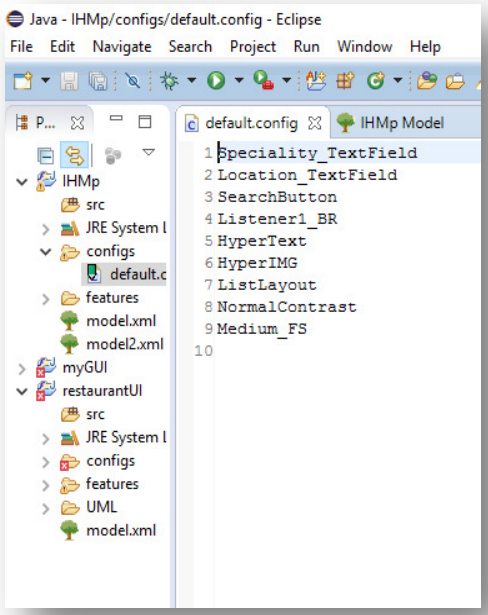


Figure 5-7 The running configuration

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

Figure 5-8 Adaptation rules (runtime phase)

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

Figure 5-9 Context datas

5.3.2 The runtime reconfiguration script

Below, we present the reconfiguration shell script:

```
#!/bin/bash
For I in 'cat contextdata'
Do
Var1='cut -f2 -d ' ' $i'
Var2='cut -f3 -d ' ' $i'
If [$var2='false'];then
For J in 'cat defaultconfig'
Do
If [$j -eq $var1];then
Sed '/$var1/d' defaultconfig
Endif
Done
Else
Exist=false
For J in 'cat defaultconfig'
Do
If [$j=$var];then
Exist=true
Endif
Done
If [$exist=false];then
Exist=true
For k in 'cat adaptationrules'
Do
Var3= 'cut -f2 -d ' ' $k'
Var4= 'cut -f3 -d ' ' $k'
```

```
If [$var3=$var4];then

Action=$var4

Endif

Done

Cat $action>>defaultconfig

done
```

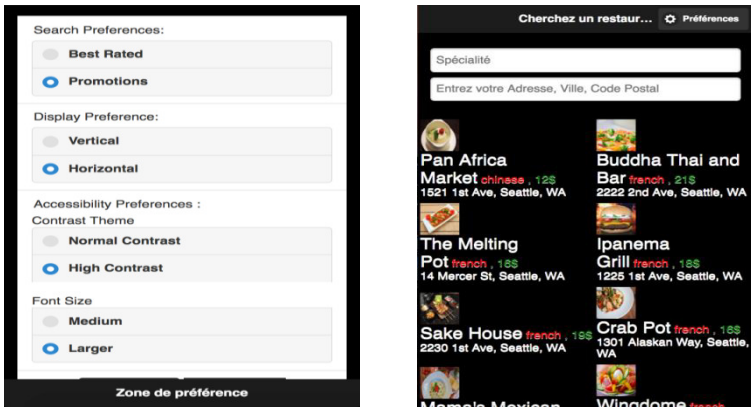
5.3.3 The runtime recomposition script

Once the new configuration is generated, it's time to recompose the new user interface. This interface is recomposed using the following script.

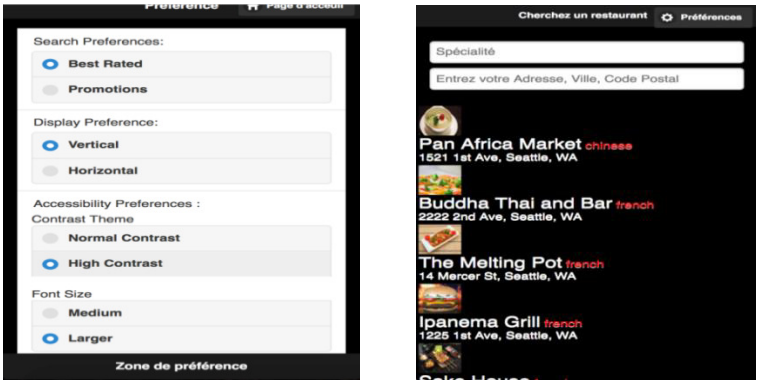
```
#!/bin/bash

 #$1 are a command parameter, $1 contains the name of the artifacts (presented
 #in form of XML files) directory and $2 contains the name of the
 #configuration file

 Touch result.xml

For I in 'cat $2'

Do

For J in 'ls $1'

Do

If [$i-eq $j];then

X='cat $j'

saxon $X merge.xslt with=result.xml >> result.xml

endif

done

done
```

5.3.4 Adapted Interfaces

Like 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 5-10 (a), the right interface is the first adaptation of the search UI. It corresponds to context 1 described as follows: the user prefers visualizing the restaurants in promotions, displayed in a horizontal way with a contrast theme and a large font size. The second interface (figure 5-10 (b)) is another adaptation for the search UI which corresponds to context 2 described as follows: the user prefers visualizing the best rated restaurants, displayed in a vertical way with a contrast theme and a large font size.



(a) Search UI adaptation – context1



(b) Search UI adaptation-Context 2

5-10 The adaptation of the search UI

5.4 Evaluation and Discussion

In our thesis so far, an approach to enable UI adaptation in DSPLs has been proposed, with details of developed tools and prototypes given. In this section, focus is given to the evaluation of the proposed approach using the prototypes presented above. The goal is to prove that the approach actually meets the goals of this thesis, and to evaluate the extent to which it scales. To validate the approach, we use a scenario-based evaluation, a scalability evaluation and a qualitative/quantitative evaluation to help assess our approach using the illustrative case study.

5.4.1 A Scenario-based Evaluation

For the scenario based evaluation and as described in chapter 3, the illustrative SPL is relative to a "search for restaurant" application. To manage the application variability, we defined two variability models: the context feature model and the UI (search 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), visual aspect (text font size, UI contrast theme) and the layout aspect (response hyperlinks displayed in the form of grid, response hyperlinks displayed vertically).

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

To implement context and UI variabilities, we used the CUI model, 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 search and context UIs.

For the other works, the illustrative SPL in Kramer's approach was about a content store case study. In his 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 used 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.

In comparison with other approaches, we note that our approach took into consideration many aspects when describing the UI variability. In addition to presentation units and UI elements, we have described the UI variability according to layout and visual appearance (text size, UI color) aspects. The design and the implementation of such features ensure the generation of an ergonomic interface; 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 reusable and more abstract.

At the runtime phase, comparing our approach with the other approaches, we note that the principal difference is about context acquisition. In our approach, the context is 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.

5.4.2 Scalability Evaluation

For scalability evaluation, we used two metrics: the generation time and the adaptation time. In table 5-1, we present these metrics and the correspondent values for our approach and Kramer's approach. Gabillon did not perform a scalability evaluation.

5.4.2.1 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 source code generation. The generation time is calculated for the context UI and the search UI. As depicted in table 5-1, the generation time of the context UI varies between 0.86s (for 1 feature) to 13m.05 s (for 10 features) while for the search UI, the generation time varies between 0.89s (for 1 feature) to 14m.57s (for 14 features). For Kramer's approach, the generation time varies between 1.11s (for 1 feature) to 23.43m (for 14 features).

5-1 Scalability evaluation results

	Metrics	
Approach	**Generation Time**	**Adaptation Time**
Kramer's Approach [41]	Min. Time=1.11s Max. Time =23.43m	Min. Time=2.44s Max. Time=63.72ms
Sboui's Approach [62,65]	**Context interface** Min.Time=0.86s/Max.Time=13.05m **Main interface** Min.Time=0.89/Max.Time=14.57m	Min. Time =2.03s Max. Time =68.332ms
Gabillon's Approach [26]	--	--

5.4.2.2 Adaptation Time

This metric measures the time taken by an adaptation cycle. This time includes the time of a new configuration + the time of recomposition of the new UI + the time of 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 2,83 s (for 1 feature) and to 68,332 ms (for 14 features). In Kramer's approach, the adaptation time is equal to 2,44s (for 1 feature) and 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.

5.4.3 The IBM CSUQ questionnaire Evaluation

5.4.3.1 The questionnaire (see appendix C)

The last evaluation is a qualitative evaluation, for that we sent an on-line questionnaire to 11 participants to evaluate their experience applying and using the Model through the two different

implementations. We relied on the IBM Computer Satisfaction Usability Questionnaire (CSUQ)[1], an empirically-validated 19-question questionnaire benefiting from a = 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=disagree, 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.

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.

5. 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 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.

[1] http://garyperlman.com/quest/quest.cgi

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.

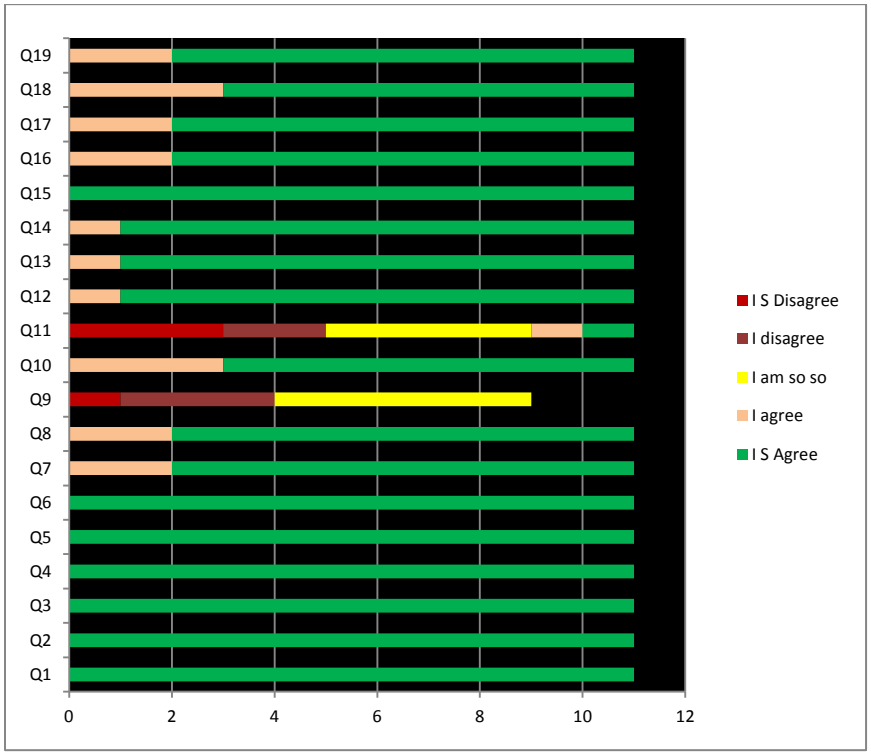
19. Q19: I am satisfied in using this model.

5.4.3.2 The Results of the Questionnaire

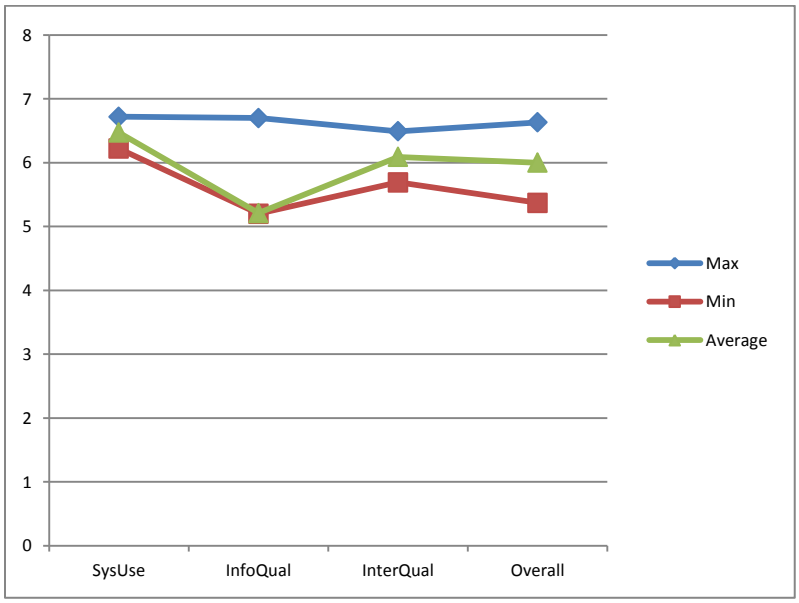
Figure 5-11 graphically depicts the distribution of the answers provided by the participants on the 19 IBM CSUQ questions. Each cumulated horizontal histogram of Figure 5-1 could be interpreted as follows: a score between 6 and 7 represented with dark green, is considered as excellent; a score of 5, 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 5-12 and 5-13 summarize the aggregated CSUQ sub-metrics reported in table 5-2. Each CSUQ questionnaire involves the calculation of four quality metrics of the system being evaluated as follows:

Table 5-2. Scores by CSUQ sub-metrics

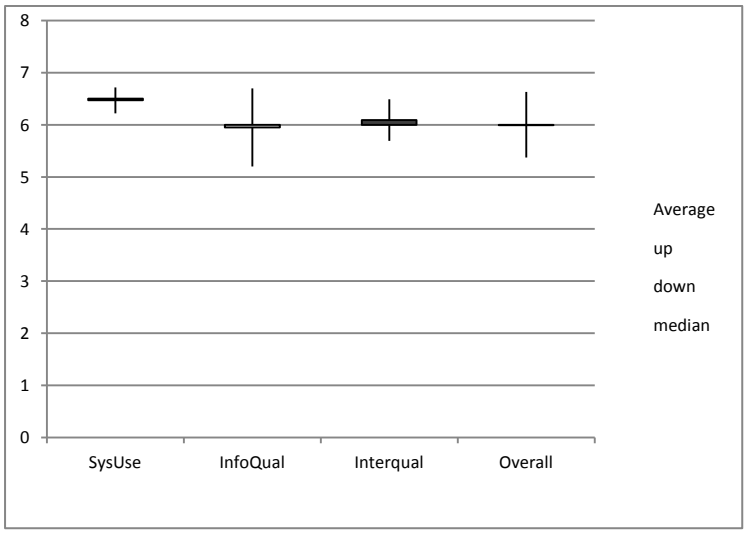
Sub-metric	Average	Mediane	Avg. Deviation	Std. Deviation
SysUse	6,47	6.5	0.84	6,5
InfoQual	4.79	6	1.05	0.8
Interqual	6,09	6	5	0.4
Overall	6	6,6	0.36	0.63



5-11. Distribution of participants' answers to the IBM CSUQ questionnaire



5-12. Aggregated scores by CSUQ sub-metrics (Min, Max, Average)



5-13. Aggregated scores by CSUQ sub-metrics(Average, Up, Down, Median)

Figure 5-11 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, μ =6, M =6, s = 0.63). The most positively evaluated sub-metric is certainly the system usefulness (Q1- Q8, μ = 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, μ = 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, μ = 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. Question 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 5-2 and Figures 5-12 and 5-13 evidences that participants perceived as "useful" the model (μ = 6.47,M = 6.5,s = 0.25) and said to be "Overall satisfied" (μ = 6,M = 6,s = 0.63).

5.5 Conclusion

In this chapter, we presented the implementation of the design phase and the runtime phase of our UI-DSPL approach. We highlighted the used tools and some developed prototypes. For development, we, mainly, used shell scripts, the XML markup language and the XSLT transformation Language. Prototypes was developed using shell scripts. The rest of tasks will be done in future works. For evaluation, three main evaluation methods were used: an SPL-based

evaluation, a scalability evaluation and a qualitative/quantitative evaluation based on the IBM/CSUQ questionnaire. In the next chapter, we conclude our dissertation and we give some perspectives.

Chapter 6. Conclusion & future works

6.1 Introduction

In this thesis, a DSPL approach for UI adaptation was presented. This chapter concludes the thesis providing an overall summary in Section 6.2. In Section 6.3, we summarize the main contributions of our work, and finally (Section 6.4), we discuss the work that will be done in the future.

6.2 Thesis Summary

In the high proliferation of smart devices, and mobile applications, user requirements are always changing. For that, using context-aware adaptive computing has become a necessity to adapt the software to user's needs. In this context, the use of Dynamic software product line (DSPL) paradigm was increased. DSPL exploits the knowledge acquired in SPLE to develop a family of systems that can be context-aware, post-deployment reconfigurable, or runtime adaptable.

Our dissertation is considered as a new contribution in the field of user interfaces adaptation. In our thesis, we have proposed a UI-DSPL approach for the development of context-adaptable UIs. Our approach includes a design phase and a runtime phase. The design phase was reserved for the development of initial UIs while the runtime phase was reserved for the UI adaptation to the context change.

6.3 Thesis Contributions

Our thesis faces four main contributions. For contributions that have been fulfilled in the design phase, we find: 1) the design and the implementation of a profiled context, 2) the combination of DSPL and MBUID artifacts in order to ensure the abstraction and the reusability of the proposed approach. For the challenges that have been fulfilled at the runtime phase, we find, 1) the proposition of a design pattern that helps to design and develop a runtime adaptation mechanism and 2) the implementation of the runtime adaptation mechanism. In the following, a brief overview on these contributions:

6.3.1 The Design and the implementation of a Profiled context

Contrary to existing UI-DSPL approaches, in which authors dealt with platform adaptable UIs, in our dissertation, we deal with interfaces adaptable to a profiled context. A profiled context is a context which is manually entered by the end user (e.g. user preferences). To be generated, a context UI is configured, at the design phase, using its feature model. For configuration, we used a default user preference. After that, we use CUI artifacts to implement the context variants and generate the context interface. At the runtime phase, new context data (user-specific preferences) are captured in order to adapt the desired interface according to the context.

6.3.2 Make the UI-DSPL approach more abstract and reusable

Another design phase contribution was the use of Model Based User Interface Design models to implement context and UI features. The use of models, instead of, components, aspect or any other implementation technology ensures the abstraction and the reuse of the UI-DSPL approach. As MBUID model, we opted for the concrete user interface model (CUI model). This later

describes the UI in terms of concrete interactors (e.g. widget, container, and layout). The feature mapping was realized manually by matching a CUI model fragment to each feature.

6.3.3 A design pattern for the runtime adaptation mechanism

This challenge was intended for the runtime phase. The runtime adaptation model defines the adaptation concepts, the relation between them and will serve as a design pattern aiming to facilitate the design and the development of the runtime adaptation mechanism.

6.3.4 A runtime adaptation mechanism to adapt user interfaces

The second runtime challenge is the adaptation engine. This later is responsible of the reconfiguration and the recomposition of the user interface at the runtime. For the mechanism components, we find the context acquisition component which acquires the context data, the adaptation script, responsible of the UI reconfiguration and recomposition according to context data and the UI generator, responsible of the generation of the adapted UI source code.

6.4 Future works

Even if the results obtained from this research are relevant, there are still several areas that have to be further developed. In the following paragraphs we discuss some of these areas as well as some future works that could lead to further improvements in the DSPL approach.

The work realized in the context of this dissertation can be continued and several areas are still open for improvement. Below, we present some of the works that would further improve our approach in the short term:

6.4.1 Short-term perspectives

Perspective 1: Optimization of the runtime UI Recomposition algorithm: a first task that can be optimized in future works is the UI re-composition. This process is used at the design phase to compose initial UIs and at the runtime phase to recompose the adapted UIs. The recomposition consists in merging the CUI model artifacts corresponding to the selected features.

Using the same recomposition algorithm at both phases seems useful. However, at the runtime phase, there is a slight difference in UI recomposition. If we have to adapt one part of the UI, the used recomposition algorithm doesn't seem to be the best solution. Because if we apply the design phase composition algorithm at the runtime phase, UI parts that don't need adaptation will be recomposed and consequently lose their values. So, the best solution will be to recompose only UI parts which need adaptation.

Perspective 2: Optimization of the runtime configuration: At the runtime, when looking for a new configuration, the reconfiguration algorithm missed to verify constraints conflicts. So, in order to optimize the runtime reconfiguration algorithm, this later has to consider the resolution of constraint conflicts.

Perspective 3: use another evaluation method to evaluate our approach (see appendix B). This evaluation will be performed by designers/developers. We are invited to answer to certain number of questions [29]. After that and following the result of the questionnaire, we can detect the gaps and fill them in our future work.

6.4.2 Long-term perspectives

Perspective 1: cloud architecture

To facilitate the modification of our approach and its extension, we propose a cloud computing architecture. The use of cloud architecture will ensure many advantages for the back side (experts/developers) and the front side (the final user). As described in figure 6-1, the cloud is shared by UI experts, SPL experts and the final user. On the one hand, SPL expert with the collaboration of UI expert can use the SPL Integrated Development Environment (IDE), UI sketching tool and any other useful environment for the development of UI-SPL process without worrying about storage or other issues. At the backend, the UI-SPL implementation environment (Eclipse IDE) needs mechanisms such as composers and solvers. At the frontend and at the runtime phase, if the final user has particular preferences, he applies them through his interface. An Application Programming Interface (API) communicates these changes to an adaptation engine that will trigger an adaptation process. This process will need mechanisms used by the Eclipse IDE (e.g.: composer, solvers) and resources resulted from the design time process (e.g.: UI, features models, configurations, assets and adaptation rules).

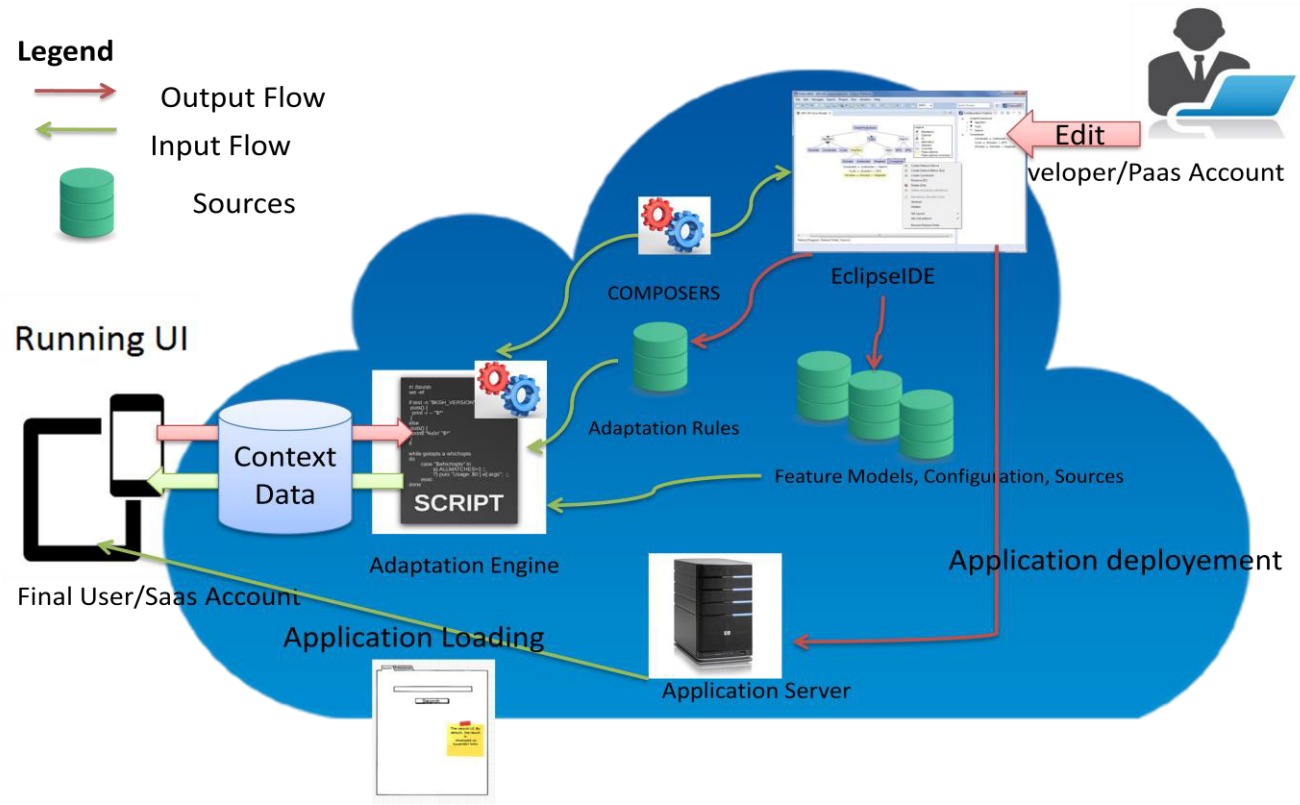


Figure 6-1 A cloud architecture

To conclude, the benefits of using cloud architecture can be summarized as follows:

- The sharing of development environment between UI expert and SPL expert/developers;
- The sharing of data storages and mechanisms (i.e: composer) between developers and final user SPL;
- The sharing of the adaptation engine between users of the running interface;
- The lightness of cloud-users;
- The Scalability of the application if we want to extend the application by adding additional functionalities or other contributors;
- The Reduction of application development and application adaptation cost.

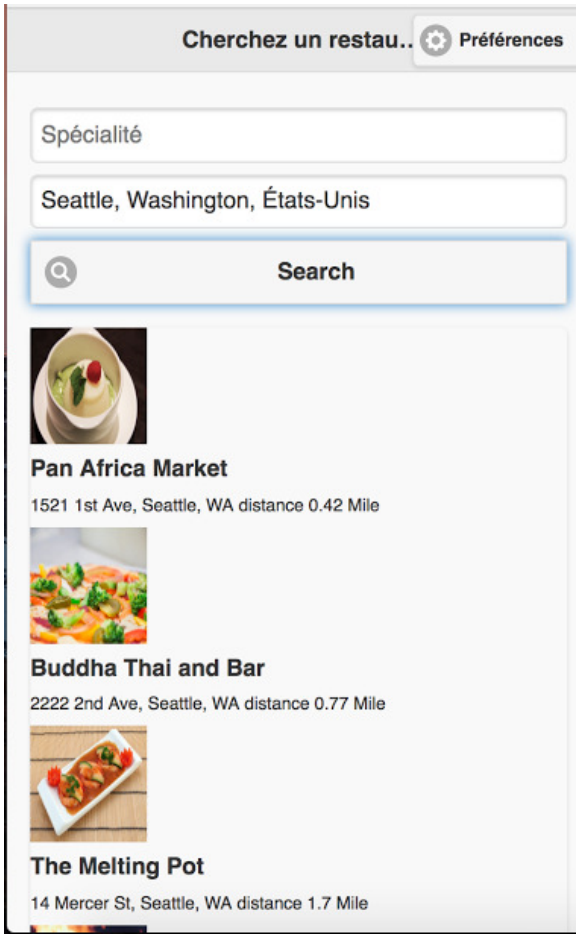
Appendix A

The XSLT File of the Generation of the Search Interface

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:fn="http://www.w3.org/2005/xpath-functions">
	<xsl:output method="html" indent="yes"/>
	<xsl:template match="/">
	<xsl:param name="file1" select="document('restaurants.xml')"/>
<!--	<xsl:result-document href="index.html">-->

	<html>
	<head>
	<link rel="stylesheet" type="text/css" href="css/thouraya.css"/>
	</head>
	<body>
	<script src="js/Thouraya.js"/>
	<div data-role="header">

		<h1> Cherchez un restaurant </h1>
		<a href="preference.php" data-icon="gear" class="ui-btn-right" data-
transition="fade">Pr&#233;f&#233;rences</a>
	</div>

	<div data-role="main" class="ui-content">
		<input type="text" placeholder="Sp&#233;cialit&#233;" id="myInputName"
onkeyup="myFunctionName()"/>
		<input type="text" placeholder="Entrez votre Adresse, Ville, Code Postal"
id="myInputAdd" onkeyup="myFunctionAdd()"/>
	</div>

	<table id="myTable">
	<xsl:for-each select="$file1//restaurant">
	<tr>
	<td>
	<img class="imagerestaurant" alt="{@name}" src="image/{@image}"/><br/>
	<span style="font-size: 18pt;"><xsl:value-of select="@name"/></span><br/>
	<xsl:value-of select="@address"/>
	</td>
	</tr>


</xsl:for-each>
```

```
            </table>

            </body>
</html>
<!--        </xsl:result-document>-->
            </xsl:template>
</xsl:stylesheet>
```

The correspondent search UI



Appendix B

The NASA-TLX Questionnaire [29]

NASA Task Load Index

Hart and Staveland's NASA Task Load Index (TLX) method assesses work load on five 7-point scales. Increments of high, medium and low estimates for each point result in 21 gradations on the scales.

Name	Task	Date

Mental Demand — How mentally demanding was the task?

Very Low — Very High

Physical Demand — How physically demanding was the task?

Very Low — Very High

Temporal Demand — How hurried or rushed was the pace of the task?

Very Low — Very High

Performance — How successful were you in accomplishing what you were asked to do?

Perfect — Failure

Effort — How hard did you have to work to accomplish your level of performance?

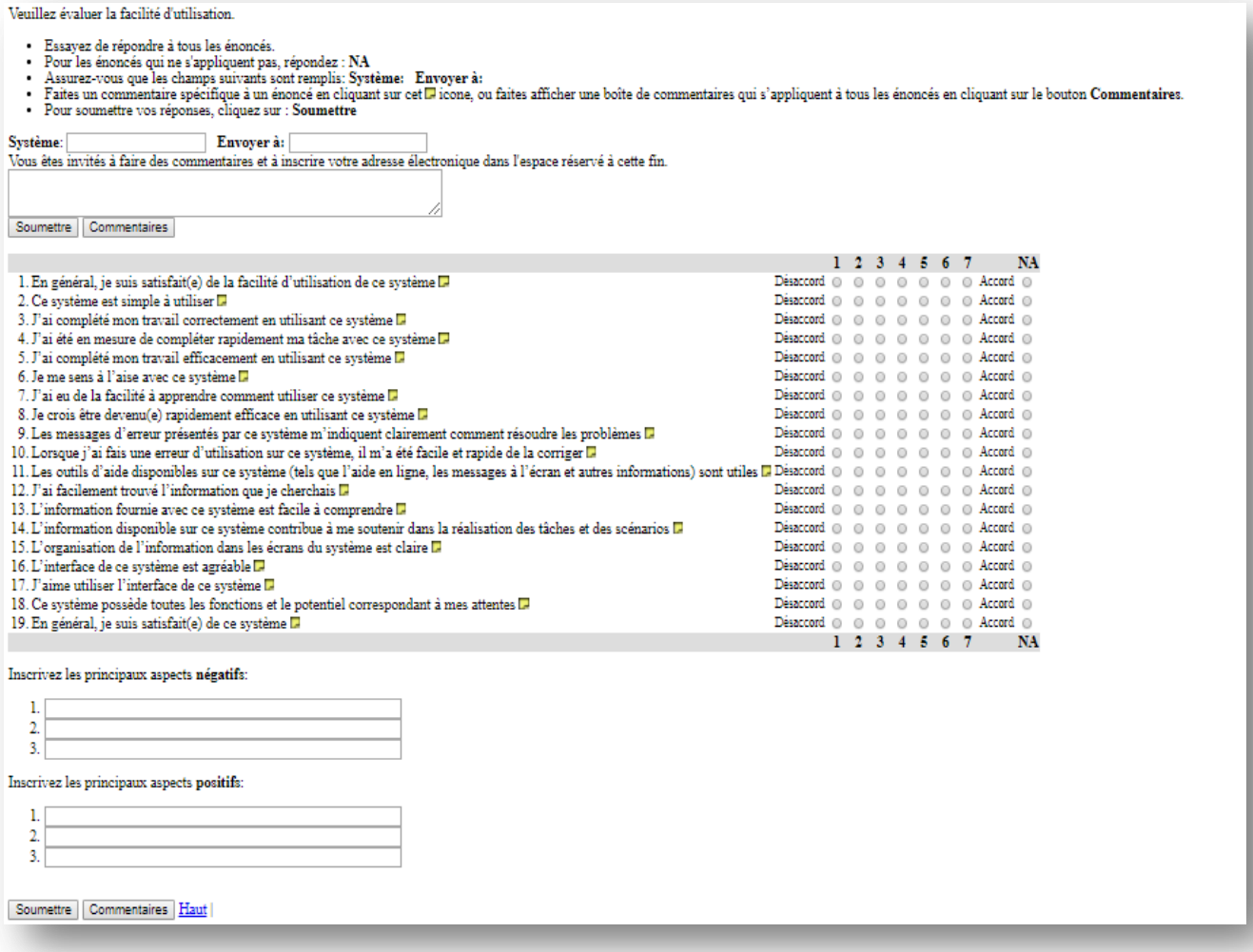
Very Low — Very High

Frustration — How insecure, discouraged, irritated, stressed, and annoyed wereyou?

Very Low — Very High

Appendix C

The IBM CSUQ Questionnaire



Veuillez évaluer la facilité d'utilisation.

- Essayez de répondre à tous les énoncés.
- Pour les énoncés qui ne s'appliquent pas, répondez : **NA**
- Assurez-vous que les champs suivants sont remplis: **Système:** **Envoyer à:**
- Faites un commentaire spécifique à un énoncé en cliquant sur cet icone, ou faites afficher une boîte de commentaires qui s'appliquent à tous les énoncés en cliquant sur le bouton **Commentaires**.
- Pour soumettre vos réponses, cliquez sur : **Soumettre**

Système: **Envoyer à:**

Vous êtes invités à faire des commentaires et à inscrire votre adresse électronique dans l'espace réservé à cette fin.

Soumettre | Commentaires

		1	2	3	4	5	6	7		NA
1. En général, je suis satisfait(e) de la facilité d'utilisation de ce système	Désaccord								Accord	
2. Ce système est simple à utiliser	Désaccord								Accord	
3. J'ai complété mon travail correctement en utilisant ce système	Désaccord								Accord	
4. J'ai été en mesure de compléter rapidement ma tâche avec ce système	Désaccord								Accord	
5. J'ai complété mon travail efficacement en utilisant ce système	Désaccord								Accord	
6. Je me sens à l'aise avec ce système	Désaccord								Accord	
7. J'ai eu de la facilité à apprendre comment utiliser ce système	Désaccord								Accord	
8. Je crois être devenu(e) rapidement efficace en utilisant ce système	Désaccord								Accord	
9. Les messages d'erreur présentés par ce système m'indiquent clairement comment résoudre les problèmes	Désaccord								Accord	
10. Lorsque j'ai fais une erreur d'utilisation sur ce système, il m'a été facile et rapide de la corriger	Désaccord								Accord	
11. Les outils d'aide disponibles sur ce système (tels que l'aide en ligne, les messages à l'écran et autres informations) sont utiles	Désaccord								Accord	
12. J'ai facilement trouvé l'information que je cherchais	Désaccord								Accord	
13. L'information fournie avec ce système est facile à comprendre	Désaccord								Accord	
14. L'information disponible sur ce système contribue à me soutenir dans la réalisation des tâches et des scénarios	Désaccord								Accord	
15. L'organisation de l'information dans les écrans du système est claire	Désaccord								Accord	
16. L'interface de ce système est agréable	Désaccord								Accord	
17. J'aime utiliser l'interface de ce système	Désaccord								Accord	
18. Ce système possède toutes les fonctions et le potentiel correspondant à mes attentes	Désaccord								Accord	
19. En général, je suis satisfait(e) de ce système	Désaccord								Accord	
		1	2	3	4	5	6	7		NA

Inscrivez les principaux aspects **négatifs**:

1.
2.
3.

Inscrivez les principaux aspects **positifs**:

1.
2.
3.

Soumettre | Commentaires | Haut |

Bibliography

[1] Abrahão, S., Iborra, E., Vanderdonckt, J.:Usability Evaluation of User Interfaces Generatedwith a Model-Driven Architecture Tool.In:MaturingUsability: Quality in Software, Interaction and Value, Law, E., Hvannberg, E., and Cockton, G. (eds.), Chapter 1. HCI Series, Vol. 10, Springer, London, pp. 3-32. DOI=http://dx.doi.org/10.1007/978-1-84628-941-5_1, 2008.

[2] Acher, M., Collet, P., Lahire, P., France, R.B.: Familiar: A domain-specific language for large scale management of feature models. Science of Computer Programming 78(6), pp. 657-681. DOI=https://doi.org/10.1016/j.scico.2012.12.004, June 2013

[3] Akiki P. A., Bandara A. K. and Yu, Y. (20. Adaptive model-driven user interface development systems. ACM Computing Surveys, 47(1), 2015.

[4] Apel, S., & Kästner, C. An overview of feature-oriented software development. Journal of Object Technology, 8(5), 49-84, 2009.

[5] Apel S., Kastner C., Liebig J., FeatureHouse: Language-Independent, Automated Software Composition, available at: http://www.infosun.fim.uni-passau.de/spl/apel/fh

[6] Arboleda, H., Romero, A., Casallas, R., Royer, J.C.: Product Derivation in a Model-Driven Software Product Line using Decision Models. In: Memorias de la XII ConferenciaIberoamericana de Software Engineering (CIbSE'2009, Medellín,pp.59-72.Accessible at http://ai2-s2-pdfs.s3.amazonaws.com/57ae/634647fcf7e610df3d106eb2a3cd0f152733.pdf , April 2009.

[7] Bacikova M., Poruban J., Analyzing stereotypes of creating graphical user interfaces, Central European Journal of Computer Science vol. 2 num. 3, pp. 300-315, 2012

[8] Batory, D. Feature models, grammars, and propositional formulas, in H. Obbink and K. Pohl (eds), Software Product Lines, Vol. 3714 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 7–2, 2015.

[9] Benavides, D., Segura, S. and Ruiz-Cort´es, A. Automated analysis of feature models 20 years later: A literature review, Inf. Syst. 35: 615–636, 2010.

[10] Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Vanderdonckt, J.: Computer-Aided Window Identification in TRIDENT. In: Proc. of 5thIFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'95 (Lillehammer, 27-29 June 1995), K. Nordbyn, P.H. Helmersen, D.J. Gilmore, S.A. Arnesen (Eds.). Chapman & Hall, London, 1995, pp. 331-336.

[11] Boucher, Q., Perrouin, G., Heymans, P.: Deriving configuration interfaces from feature models: A vision paper. In: Proceedings of the 6thInternational Workshop on Variability Modeling of Software-Intensive Systems (VaMOS'2012, Leipzig). ACM Press, New York, 2012, pp. 37-44. DOI=https://doi.org/10.1145/2110147.2110152, January 2012.

[12] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J. : AUnifying Reference Framework for Multi-Target User Interfaces.Interactingwith Computers 15(3), June 2003, pp. 289-308.

[13] Cantera Fonseca, J.M. (Ed.), González Calleros, J.M., Meixner, G., Paternò, F., Pullmann, J., Raggett, D. Schwabe, J. Vanderdonckt, Model-Based UI XG Final Report, W3C Incubator Group Report, Accessible at http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/, May 2010.

[14] Capilla, R., Bosch, J., Trinidad, P., Ruiz-Cortés, A., Hinchey, M.: An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry. Journal of Systems and Software 91, pp. 3-23. DOI=https://doi.org/10.1016/j.jss.2013.12.038, May 2014.

[15] Capilla, R., Bosch, J.: Dynamic Variability Management Supporting Operational Modes of a Power Plant Product Line. In: Proceedings of the 6thInternational Workshop on Variability Modeling of Software-Intensive Systems (VaMOS'2012, Leipzig, January 25-27, 2012). ACM Press, New York, 2012, pp. 49-56. DOI=https://doi.org/10.1145/2866614.2866621

[16] Cerny, T., Cemus, K., Donahoo, M. J., & Song, E. Aspect-driven, data-reflective and context-aware user interface, 2013.

[17] Clements, P., Northrop, L.: Software Product Lines. Addison-Wesley, Boston, 2002.

[18] Common Variability Language, omgwiki.org, retrieved November 2, 2016 from http://www.omgwiki.org/variability/doku.php?id=cvl_tool_from_sintef

[19] Czarnecki K., "Overview of generative software development", In Unconventional Programming Paradigms (pp. 326-341). Springer Berlin Heidelberg. 2005.

[20] 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

[21] Eclipse Modeling Framework, available at: http://www.eclipse.org/modeling/emf

[22] Eleutério, J. D. A. S., & Rubira, C. M. F. A Comparative Study of Dynamic Software Product Line Solutions for Building Self-Adaptive Systems, 2017

[23] Feature Modeling Plug-in, Generative Software Development Lab. Retrieved November 16, 2016, from http://gsd.uwaterloo.ca/fmp

[24] Feigenspan, J., K¨astner, C., Frisch, M., Dachselt, R. and Apel, S. Visual support for understanding product lines, Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension, ICPC '10, IEEE Computer Society, Washington, DC, USA, pp. 34–35, 2010.

[25] Fleury, M., Reverbel, F: The JBoss extensible server. In: Proceedings of Middleware, Markus Endler and Douglas Schmidt (Eds.). Lecture Notes in Computer Science, Vol. 2672. Springer,Berlin, 2003, pp. 344–373. DOI=http://dx.doi.org/10.1007/3-540-44892-6_18, 2003.

[26] Gabillon, Y., Biri, N., Otjacques, B.: Designing an adaptive user interface according to software product line engineering. In: Proceedings of 8thInternational Conference on Advances in Computer-Human Interactions(ACHI'2015, Lisbon, February 22-27, 2015). International Academy, Research, and Industry Association (IARIA), 2015, pp. 86-91. Accessible at

[27] Gajos, K. Z., Weld, D. S., & Wobbrock, J. O. Automatically generating personalized user interfaces with Supple. Artificial Intelligence, 174(12-13), 910-950. 2010.

[28] Garcés, K., Parra, C., Arboleda, H., Yie, A.,Casallas, R.: Variability management in a model-driven software product line.AvancesenSistemas e Informática**4**(2),2007. Accessible at http://www.bdigital.unal.edu.co/15155/

[29] Gawron V. J., Human Performance Measures Handbook, "Nasa TLX questionnaire", available at: http://theses.univ lyon2.fr/documents/getpart.php?id=lyon2.2010.maincent_a&part=365749, 2000

[30] Genaro Motti, V., Vanderdonckt, J.: A Computational Framework for Context-aware Adaptation of User Interfaces: In: Proceedings of 7thInt. Conf. on Research Challenges in Information Science (RCIS'2013, Paris, May 29-31, 2013). IEEE Computer Society, Los Angeles, 2013, pp. 1-12.

[31] Gomaa, H., Hussein, M.: Dynamic software reconfiguration in software product families. In: Proceedings of International Workshop on Software Product-Family Engineering (PFE'2003, Siena, November 4-6, 2003). Lecture Notes in Computer Science, vol. 3014.Springer, Berlin, 2003, pp. 435–444. DOI=http://dx.doi.org/10.1007/978-3-540-24667-1_33

[32] González Calleros J. M., Meixner G., Paternò F., Pullmann J., Raggett D., Schwabe D., Vanderdonckt J., "Model-Based UI XG Final Report", May 2010, available at: https://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/

[33] Grislin M., Kolski C., « Evaluation des interfaces homme-machine lors du développement de système interactif ». *Technique et Science Informatiques TSI*, 3, pp. 265-296, 1996.

[34] Guerrero, J., Vanderdonckt, J., Gonzalez, J.:FlowiXML: a Step towards Designing Workflow Management Systems.Int. Journal of Web Engineering and Technology**4**(2), 2008, pp. 163-182. DOI=https://doi.org/10.1504/IJWET.2008.018096

[35] Hariri, M-A., Lepreux, S., Tabary, D., Kolski, C.. : Principes et étude de cas d'adaptation d'IHM dans les SI en fonction du contexte d'interaction de l'utilisateur. Ingénierie des Systèmes d'Information (ISI), Networking and Information Systems, 14, pp. 141--162 (2009)

[36] Heidenreich F., FeatureMapper mapping features to models, available at: http://featuremapper.org, lastupdate, September 2013.

[37] Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., Vanderdonckt, J.: Human-Centered Engineering with the User Interface Markup Language.In: Human-Centered Software Engineering, Seffah, A., Vanderdonckt, J., Desmarais, M. (eds.), Chapter 7. HCI Series, Springer, London, 2009, pp. 141-173. DOI=http://dx.doi.org/10.1007/978-1-84800-907-3_7

[38] Hubaux, A., Acher, M., Tun, T.T., Heymans, P., Collet, P.,Lahire, P.: Separating concerns in feature models: Retrospective and support for multi-views. In: Domain Engineering:Product Lines, Languages, and Conceptual Models, Reinhartz-Berger, I.,

Sturm, A., Clark, T., Cohen, S., Bettin, J. (Eds.). Springer, Berlin, 2013, pp. 3-28. DOI=http://dx.doi.org/10.1007/978-3-642-36654-3_1

[39] Kastner, C., Apel, S. and Kuhlemann, M. (2009). A model of refactoring physically and virtually separated features, Proceedings of the Eighth International Conference on Generative Programming and Component Engineering, GPCE '09, ACM, New York, NY, USA, pp. 157–166

[40] Kramer, D.M.: Unified GUI adaptation in dynamic software product lines. Doctoral dissertation, University of West London, London, August 2005. Accessible at http://repository.uwl.ac.uk/1270/

[41] Kramer, D., Oussena, S., Komisarczuk, Clark, T.: Using document-oriented GUIs in dynamic software product lines. ACM SIGPLAN Notices **49**(3), March 2014, pp. 85-94. DOI=https://doi.org/10.1145/2637365.2517214

[42] Logre, I., Mosser, S., Collet, P., Riveilli, M.: Sensor data visualisation : a composition-based approach to support domain variability. In: Proceedings of the 10th European Conference on Modelling Foundations and Applications (ECMFA'2014, York, July 21-25, 2014). Lecture Notes in Computer Science, Volume 8569. Springer, Berlin, 2014, pp. 101-116. DOI=https://doi.org/10.1007/978-3-319-09195-2_7

[43] López-Jaquero, V., Vanderdonckt, J., Montero, F., & González, P. (2008). Towards an Extended Model of User Interface Adaptation: The I satine Framework. In Engineering Interactive Systems (pp. 374-392). Springer, Berlin, Heidelberg.

[44] Lutteroth C., Weber G., Modular Specification of GUI Layout Using Constraints. In: Proceedings of the 19th Australian Conference on Software Engineering. Washington, DC, USA : IEEE Comp. Soc., 2008, pp. 300–309

[45] Meixner, G., Paterno, F., & Vanderdonckt, J. Past, present, and future of model-based user interface development. i-com 10 (3): 2-11, 2011.

[46] Mezhoudi N., 2013. User interface adaptation based on user feedback and machine learning. In Proceedings of the companion publication of the 2013 international conference on Intelligent user interfaces companion (IUI '13 Companion). ACM, New York, NY, USA, 25-28. DOI=10.1145/2451176.2451184 http://doi.acm.org/10.1145/2451176.2451184

[47] Mostefaoui, G. K., Pasquier-Rocha, J. & Brezillon, P. 2004. Context-aware computing: a guide for the pervasive computing community. IEEE/ACS International Conference on Pervasive Services. IEEE Computer Society, Lebanon, 39–48.

[48] Muller, J.: Generating graphical user interfaces for software product lines: A constraint-based approach. In: Alt et al. (Eds.), Tagungsband 15. InteruniversitäresDoktorandenseminarWirtschaftsinformatik der Universitäten Chemnitz, Dresden, Freiberg, Halle-Wittenberg, Jena und Leipzig, Leipzig,2011, pp. 64-71. Accessible at https://pdfs.semanticscholar.org/186d/7f0907852cbaaf798513ea1f2e347a63b342.pdf

[49] Object Management Group, IFML: the Interaction Flow Modeling Language, 2015.Accessible at: http://www.ifml.org/

[50] Object Management Group, Query/View/Transformation. Accessible at http://www.omg.org/spec/QVT/

[51] Parra, C. (2011). Towards dynamic software product lines: Unifying design and runtime adaptations (Doctoral dissertation, INRIA Lille).

[52] Paternò, F., Santoro, C., Spano, L.D.: Concur Task Trees (CTT). Accessible at: https://www.w3.org/2012/02/ctt/, 2012

[53] Pleuss, A., Hauptmann, B., Dhungana, D., Botterweck, G.: User interface engineering for software product lines: the dilemma between automation and usability. In: Proceedings of the 4thACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS'2012, Copenhagen, June 25-26, 2012). ACM Press, New York, 2012, pp. 25-34. DOI=https://doi.org/10.1145/2305484.2305491

[54] Pleuss, A., Hauptmann, B., Keunecke, M., Botterweck, G.: A case study on variability in user interfaces.In: Proceedings of the 16thInternational Software Product Line Conference (SPLC'2012, Salvador, September 2-7, 2012), volume 1. ACM Press, New York, 2012, pp. 6–10. DOI=https://doi.org/10.1145/2362536.2362542

[55] 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 (EICS'2013, London, June 24-27, 2013). ACM Press, New York, 2013, pp. 13-22. DOI=https://doi.org/10.1145/2494603.2480298

[56] Pohl, K., Boeckle, G., van der Linden, F.: Software Product Line Engineering. Springer, Berlin, 2005.

[57] Pure systems, pure-systems.Com, retrieved Octobe 22, from https://www.pure-systems.com/products/pure-variants-9.html

[58] Quinton, C., Mosser, S., Parra, C., Duchien, L.: Using Multiple Feature Models to Design Applications for Mobile Phones. In: Proceedings of the 15th International Software Product Line Conference (SPLC'2011, Munich, August 21-26, 2011), vol. 2, article No. 23. ACM Press, New York, 2011. DOI=https://doi.org/10.1145/2019136.2019162

[59] Rosenmüller, M., Siegmund, N., Pukall, M., Apel, S.: Tailoring dynamic software product lines. ACM SIGPLAN Notices **47**(3), March 2012, pp. 3-12. DOI=https://doi.org/10.1145/2189751.2047866

[60] Samaan, K., Tarpin-Bernard, F.: Task models and Interaction models in a Multiple User Interfaces generation process. In Proceedings of 3rd International Workshop on TAsk MOdels andDIAgrams for user interface design TAMODIA'2004. Prague, Check Republic, November, ACM, pp. 137-144(2004)

[61] Sboui, T., & Ayed, M. B. Generative Software Development Techniques of User Interface: Survey and Open Issues. *International Journal of Computer Science and Information Security*, *14*(7), 824, 2016

[62] Sboui T., A DSPL approach for the development of context-adaptable user interfaces, RCIS conference, 2017

[63] Sboui, T., Ben Ayed, M., Alimi, M. A., A Meta-model for Run Time Adaptation in a UI-DSPL process, BHCI conference, 2017

[64] Sboui, T., Ayed, M. B., & Alimi, A. M. (2017). Addressing Context-Awareness in User Interface Software Product Lines (UI-SPL) Approaches. In *Human Centered Software Product Lines* (pp. 131-152). Springer, Cham.

[65] Sboui, Thouraya, Mounir Ben Ayed, and Adel M. Alimi. "A UI-DSPL approach for the development of Context-adaptable user interfaces." IEEE Access 2017.

[66] Schlee, M., Vanderdonckt, J.: Generative Programming of Graphical User Interfaces. In: Proc. of 7th Int. Working Conference on Advanced Visual Interfaces (AVI'2004, Gallipoli, May 25-28, 2004). ACM Press, New York, 2004, pp. 403-406. DOI=https://doi.org/10.1145/989863.989936

[67] Schmidt, D.C.: Model-driven engineering. IEEE Computer 39(2), February 2006, pp. 25-31. DOI=https://doi.org/10.1109/MC.2006.58

[68] Software Product Lines Online Tools, splot-research.org. retrieved October 22, 2016, from http://www.splot-research.org/

[69] Software&SystemsProcessEngineering Metamodel™ Specification, available at: http://www.omg.org/spec/SPEM/2.0/

[70] Sottet, JS., Calvary, G., Favre, JM.: Mapping Model: A First Step to Ensure Usability for sustaining User Interface Plasticity. In: Proceedings of the MoDELS'06 Workshop on Model Driven Development of Advanced User Interfaces (2006)

[71] Sottet, J.S., Vagner, A., GarcíaFrey, A.: Model Transformation Configuration and Variability Management for User Interface Design. In: Proceedings of 3rdInternational Conference on Model-Driven Engineering and Software Development (MODELSWARD'2015, Angers, February 9-11, 2015). Communications in Computer and Information Science, vol. 580. Springer, Berlin, 2015, pp. 390-404. DOI=http://dx.doi.org/10.1007/978-3-319-27869-8_23

[72] Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., & Leich, T. (2014). FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming*, *79*, 70-85.

[73] Ubayashi, N., Nakajima, S.: Separation of Context Concerns— Applying Aspect Orientation to VDM. In Talk at the 2nd Overture Workshop, FM, Vol. 6, 2006. Accessible at https://www.yumpu.com/en/document/view/27958389/applying-aspect-orientation-to-vdm-wiki

[74] UsiXML User Interface eXtensible Markup Language: http://www.w3.org/2005/Incubator/model-basedui/wiki/UsiXML

[75] Vanderdonckt, J.: A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In: Proc. of 17thConf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005).O. Pastor & J. Falcão e Cunha (Eds.). Lecture Notes in Computer Science, Vol. 3520. Springer, Berlin, 2005, pp. 16-31.

[76] Voelter, M. and Groher, I. (2007). Product line implementation using aspect-oriented and model-driven software development, Proceedings of the 11th International Software Product Line Conference, SPLC '07, IEEE Computer Society, Washington, DC, USA, pp. 233–242

[77] XSLT Stylesheets Useful things and other jokes, available at: http://web.archive.org/web/20160809092524/http://www2.informatik.hu-berlin.de/~obecker/XSLT/, 2002

Tunisian Republic
Ministry of Higher Education and Scientific Research

University of Sfax
High School of Engineering of Sfax

Doctoral School of Sciences and Technologies

PhD Thesis
Computer Systems Engineering

N° d'ordre: 200?– ??nn nn

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

Thouraya SBOUI

الخلاصة: تقدم هذه الأطروحة مساهمة جديدة في مجال تكييف الواجهات بين الإنسان والآلة. هنا، يتم استخدام خط منتج ديناميكي لتوليد عائلة من واجهات مصممة خصيصا لتفضيلات المستخدم. يتضمن خط منتجاتنا الديناميكي مرحلتين، مرحلة تصميم مخصصة لتطوير الواجهات الأولية ومرحلة تنفيذ مخصصة لتكييف الواجهة. وتحدد كل مرحلة من مراحل نهجنا مجموعة من المساهمات الموصوفة والمنفذة في التقرير. ولتحقيق هذا النهج، تم استخدام ثلاثة أساليب للتقييم، تقييم يستند إلى سيناريو خط الانتاج، تقييم قابلية للتطوير، وتقييم نهائي يستند إلى استبيان .IBM CSUQ

Résumé : Cette thèse présente un nouvel apport dans le domaine de l'adaptation des interfaces Homme-Machine. Dans la présente, on utilise une ligne de produits dynamique pour la génération d'une famille d'interfaces adaptées aux préférences de l'utilisateur. Notre ligne de produit dynamique renferme deux phases, une phase de conception dédiée au développement des interfaces initiales et une phase d'exécution réservée à l'adaptation de l'interface. Chaque phase de notre approche définit un ensemble de contributions qui sont décrites et implémentées au sein du rapport. Pour valider l'approche, trois méthodes d'évaluation ont été utilisées, une évaluation basée sur un scénario SPL, une scalability-évaluation et une dernière évaluation basée sur le questionnaire IBM CSUQ.

Abstract: This thesis presents a new contribution in the field of the adaptation of human-machine interfaces. Here, a dynamic product line is used to generate a family of interfaces tailored to the user's preferences. Our dynamic product line includes two phases, a design phase dedicated to the development of the initial interfaces and an execution phase dedicated to the adaptation of the interface. Each phase of our approach defines a set of contributions that are described and implemented within the report. To validate the approach, three evaluation methods were used, an evaluation based on an SPL scenario, a scalability-evaluation and a final evaluation based on the IBM CSUQ questionnaire.

المفاتيح: التكيف حسب السياق ,تكيف واجهة المستخدم ,خط إنتاج ديناميكي.

Mots clés: Adaptation au contexte, approche DSPL, adaptation de l'interface utilisateur.

Key-words: Context-adaptation, DSPL approach, UI Adaptation