Variability Management and Assessment for User Interface Design

Jabier F Martinez, Jean-Sébastien Sottet, Alfonso García Frey, Tewfik Ziadi, Tegawendé F. Bissyandé, Jean Vanderdonckt, Jacques Klein, Yves Le Traon

Chapter 3
Variability Management and Assessment for User Interface Design

Jabier Martinez, Jean-Sébastien Sottet, Alfonso García Frey, Tewfik Ziadi, Tegawendé Bissyandé, Jean Vanderdonckt, Jacques Klein, and Yves Le Traon

Abstract User Interface (UI) design remains an open, wicked, complex and multi-faceted problem, owing to the ever increasing variability of design options resulting from multiple contexts of use, i.e., various end-users, heterogeneous devices and computing platforms, as well as their varying environments. Designing multiple UIs for multiple contexts of use inevitably requires an ever growing amount of time and resources that not all organizations are able to afford. Moreover, UI design choices stand on end-users' needs elicitation, which are recognized to be difficult to evaluate precisely upfront and which require iterative design cycles. All this complex variability should be managed efficiently to maintain time and resources to an acceptable level. To address these challenges, this article proposes a variability management approach integrated into a UI rapid prototyping process, which involves the combination of Model-Driven Engineering, Software Product Lines and Interactive Genetic Algorithms.

J. Martinez (✉) • T. Ziadi
Sorbonne University, UPMC Univ Paris 06, CNRS, Paris, France
e-mail: jabier.martinez@lip6.fr; tewfik.ziadi@lip6.fr

J.-S. Sottet
Luxembourg Institute of Science and Technology (LIST), 5 Avenue des Hauts-Fourneaux, Esch/Alzette, Luxembourg
e-mail: jean-sebastien.sottet@list.lu

A.G. Frey
Yotako S.A., 9 Avenue des Hauts-Fourneaux, Esch/Alzette, Luxembourg
e-mail: alfonso@yotako.io

T. Bissyandé • J. Klein • Y. Le Traon
Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Luxembourg City, Luxembourg
e-mail: tegawende.bissyande@uni.lu; jacques.klein@uni.lu; yves.letraon@uni.lu

J. Vanderdonckt
Louvain School of Management, Université catholique de Louvain, Place des Doyens, 1, 1348, Louvain-la-Neuve, Belgique
e-mail: jean.vanderdonckt@uclouvain.be

3.1 Introduction

The development life cycle of User Interfaces (UIs) often requires producing several versions of the same UI intended for different targets. This development may span from different versions targeting different contexts of use to a single version exhibiting a multi-layer UI [39] offering different levels of sophistication depending on the end-user's expertise. Interaction design is understood as a complex and multi-faceted problem [8] that is always open [45] (new end-user requirements may appear while designing), iterative (there is no optimal solution at first glance), and intrinsically incomplete (not all end-user requirements are elicited from the beginning). When designing interaction, variability is manifold depending on the context of use [10]: variability of end-users, of their devices, computing platforms, and of environments in which they are working. Moreover, end-user requirements are difficult to evaluate precisely upfront in UI design processes. Therefore, the main UI design processes, such as User-Centred Design [15], implement an iterative design cycle in which a UI variant is produced, tested on end-users, and their feedback is integrated into design artifacts (e.g., part of the UI, requirements, etc.). Since these processes are mostly based on trial and error, some parts of the UI have to be re-developed many times to fulfill all the different user requirements. When different UIs are produced for various contexts of use, their respective usability to be achieved is also varying, thus posing the problem of multi-target usability evaluation [1]. Moreover, these UI design processes involve multiple stakeholders playing different roles (e.g., software developers, UI/User eXperience designers, business analysts, end-users) that demand a great amount of time to reach consensus. UI variability has thus a significant impact on the design, development, and maintenance costs of the UI, thus redistributing the total cost of ownership of a UI depending on these parameters.

To overcome variability issues in software engineering, researchers have successfully relied on the paradigm of Software Product Lines (SPLs) [12]. The SPL paradigm allows to manage variability by producing a family of related product configurations (thus leading to product variants) for a given domain. Indeed, the SPL paradigm proposes the identification of common and variable sets of features, to foster software reuse in the configuration of new products [34].

Model-Driven Engineering (MDE) has already been used to effectively support the UI design process [40]. According to [5, 13], SPL and MDE are complementary and can be combined in a unified process that aggregates the advantages of both methods without suffering too much from their respective shortcomings.

A MDE-SPL approach could produce rapidly many variants of the same UI. However, some of these variants could not satisfy their respective level of usability: a same UI deployed on different computing platforms could have different levels of usability [1]. Testing a very large collection of similar variants could be exhausting for the end-users and may lead to inappropriate results. As such, an innovative way to evaluate only a relevant portion of the variant needs to be defined. This solution could provide a reasonably good (but not necessarily optimal) variant that satisfy the end-user requirements elicited for a set of contexts of use.

This article proposes an approach to manage UI variability jointly based on MDE and SPL. Our approach relies on model transformations that support the expression of the variability. This approach enables the separation of concerns of the different stakeholders when expressing the UI variability and their design choices (UI configurations). A Multiple Feature Model approach is considered in which each feature model represents a particular concern allowing, if needed, each stakeholder to work independently. A partial and staged configuration process [14] is proposed in which a partial UI configuration is produced that can be refined by all stakeholders including the end-user feedback. Once the UI is produced, an innovative mean for evaluating the variants with end-users allowing to select the best products is presented. These concepts will be exemplified with a concrete example of UI variability.

The chapter is structured as follows: First, in Sect. 3.2, we introduce the related work on the three domains: feature modelling and configuration, user interface variability and variant testing. Next, we introduce our UI-SPL approach (Sect. 3.3) that help in deriving many products from initial design models and variability models. Then, in Sect. 3.4, we explain the assessment of the produced UI variants using a genetic approach. Before concluding the chapter, we illustrate (Sect. 3.5) and evaluate (Sect. 3.6) our approach based on a case study: evaluating the UI variants of a contact list.

3.2 Related Work

3.2.1 Feature Modelling

Feature models (FM) [34] are popular SPL assets that describe both variability and commonalities of a system. They express, through some defined operators, the decomposition of a product related features. The feature diagram notation used in this article is explained in Fig. 3.1. The E-Shop FM consists of a mandatory feature "catalogue", two possible payment methods from which one or both could be selected, an exclusive alternative of security levels and an optional search feature. FM constraints can be defined. In this case "credit card" implies a high level of security.

Features composing a FM depict different parts of a system without any clear separation of concerns. The absence of feature types makes these models popular as there are no limits for the expression of design artifacts. But at the same time, [9] have demonstrated that depicting information in a single FM leads to feature redundancies due to the tree structure. As a result, separation of variability concerns into multiple FMs seems to be crucial for understanding [28] and manipulating [2] the many different faces of variability. Each of these FM focuses on a viewpoint on variability which makes easier to handle variability for each stakeholder. An early example of this approach is MiniAba [38], which models UI adaptation based on

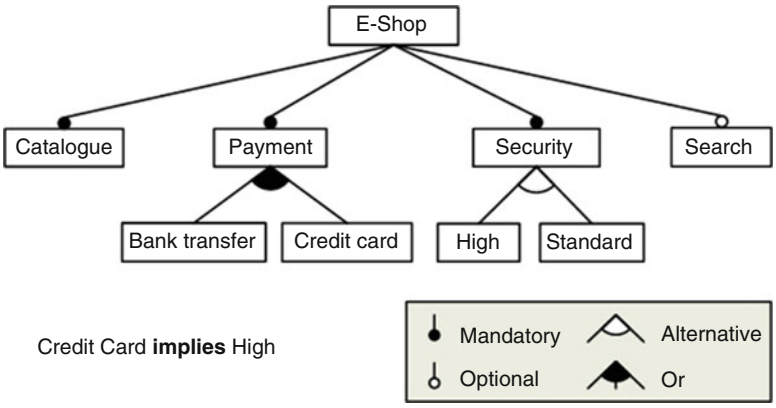


Fig. 3.1 Feature model from an E-shop (Source: Wikimedia commons)

a FM: the complete UI configuration is represented by a FM tree, whose branches and/or leaves may be edited or removed in order to produce an instantiated FM. This FM then gives rise to a project configuration that will automatically generate a corresponding adapted UI. The big win is that, when one changes any feature in the FM, it is automatically reflected in the generated UI after re-compiling.

3.2.2 SPL Configuration

The configuration process is an important task of SPL management: producing a particular product variant based on a selection of features to fit the end-users' requirements. In this context, a configuration is hereby defined as a specific combination of FM features satisfying all the FM constraints. When designers and developers configure a system according to requirements, the enforcement of FM constraints can limit them in their design choices [48]. Moreover, the separation of the variability in multiple FMs is also a source of complexity due to many dependencies across FMs. The fusion of all FMs into one for configuration purposes seems to solve this issue but results in a large FM that mixes different facets: this may lead to invalid configurations and thus invalid or inefficient products. Some solutions exist to overcome these problems. The work by [35] proposes an implementation of a configuration composition system defining a step-by-step configuration [14] using partial configurations [7]. Thus, some portion of a FM can be configured independently, without considering all the constraints (coming from other configurations) at configuration time. Then, constraints amongst configurations may be solved by implementing consistency transformations [2].

3.2.3 Model-Driven User Interfaces Variability

Model-Driven UI calls for specific models and abstraction. These models address the flow of UIs, which could range from a mono-path flow to multi-path workflow [19], the domain elements manipulated during the interaction, the models of expected UI quality, the layout, the graphical rendering, etc. In addition, each model corresponds to a standard level of abstraction as identified in the CAMELEON Reference Framework (CRF) [10]. The CRF aims at providing a unified view on modelling and adaptation of UIs. In the CRF, each level of abstraction is a potential source of variability. Modifying any model fragment at any specific level of abstraction induces a specific adaptation of the UI.

For instance, the task model, considered in CRF as the topmost model, depicts the interaction between the user and the features offered by the software in a way that is computing-independent. Adding or removing a task results in modifying the software features. Considering this, we can assume that there is a direct link between classical feature modelling and task modelling such as presented in [33]. In this work, a task model is derived from an initial FM. However the authors did not go any further in describing the variability related to interaction and UI (e.g., graphical components, behavior).

In [18], the authors present an integrated vision of functional and interaction concerns into a single FM. This approach is certainly going a step further by representing variability at the different abstraction levels of the CRF. However, this approach has several drawbacks. On the one hand, this approach derives functional variability only from the task model, limiting the functional variability of the software. On the other hand, all the UI variations are mixed into a single all encompassing FM which blurs the various aspects for comprehension and configuration [28].

Finally, Martinez et al. [29] presented a preliminary experience on the usage of multiple FMs for web systems. This work showed the feasibility of using multiple FMs and the possibility to define a process around it. It implements FMs for a web system, interaction scenario, a user model (user impairments), and device. However, this approach does not consider the peculiarities of UI design models and their variability.

A few works in SPL for UI have been published. A large part is dedicated to the main variability depiction (using FMs) but they do not directly address the configuration management. Configuration is a particular issue when considering end-user related requirements which may be fuzzily defined.

3.2.4 Testing Many Variants

Selecting optimal SPL product variants based on some criteria has been studied in SPL Engineering (SPLE). To achieve this, it is a common practice to enhance the

variability model by what is referred to as quality attributes [6, 11, 23, 25]. Assessing the quality of produced variants requires to deal with a potential large set of similar UI to be evaluated and with subjectivity (i.e., considering users feedback could also encompass some aesthetic aspects).

Genetic algorithms have been used for guiding the analysis of configuration space [17, 36]. A key operator for evolutionary genetic algorithms is the *fitness function* representing the requirements to adapt to. In other words, it forms the basis for selection and it defines what improvement means [16]. In our case, the fitness function is based on user feedback which is a manual process in opposition to automatically calculated fitness functions (e.g., the sum of the cost of the features). Because we deal with these user assessments as part of the search in the configuration space, we leverage Interactive Genetic Algorithms (IGA) where humans are responsible to interactively set the fitness [16, 47].

3.3 UI-SPL Approach

Model-driven UI design is a multi-stakeholder process [21, 22] where each model – representing a particular sub-domain of UI engineering – is manipulated by specific stakeholders. For instance, the choice of graphical widgets to be used (e.g. radio button, drop-down list, etc.) is done by a graphical designer, sometimes in collaboration with the usability expert and/or the client. Our model-driven UI design approach [40] relies on a revised version of the CRF framework (Fig. 3.2). It consists of two base meta-models, the Domain meta-model -representing the domain elements manipulated by the application as provided by classical domain analyst- and the Interaction Flow Model (IFM) [31]. From these models we derive the Concrete User Interface model (CUI) which depicts the application "pages" and their content (i.e., widgets) as well as the navigation between pages. The CUI meta-model aims at being independent of the final implementation of any graphical element. Finally, the obtained CUI model is transformed into an Implementation Specific Model (ISM) that takes into account platform details (here platform refers to UI tool-kits such as HTML/jQuery, Android GUI, etc.). Finally, a Model-to-Text (M2T) transformation generates the code according to the ISM. This separation allows for separate evolution of CUI metamodel and implementation specific metamodel and code generation.

We propose a multiple feature models (multi-FM) approach (see Sect. 3.3.1) to describe the various facets of UI variability (e.g., UI layout, graphical elements, etc.). In a second phase, (see Sect. 3.4) we introduce our specific view on configuration on this multi-FM and its implementation in our model-driven UI design approach (see Sect. 3.3.2).

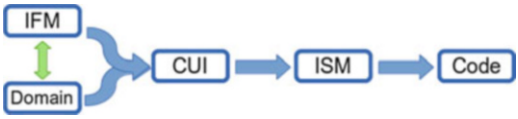


Fig. 3.2 Model-driven UI design process

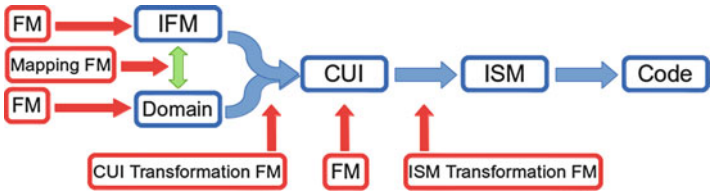


Fig. 3.3 Variability models (FM) coverage on our UI modeling framework

3.3.1 Multi-FM Approach

Classical FM approaches combine different functional features [33]. In the specific context of UI design, we propose to rely on a similar approach for managing variability of each UI design concern. UI variability is thus decomposed into FMs (Fig. 3.3). Each of these FMs is related either to a model, a meta-model, a mapping or a transformation depending on the nature of the information it conveys.

- *Models*: Three FMs in Fig. 3.3 manage variations at the model level (IFM, Domain and CUI). The FM responsible for the Domain configuration can be used to express alternatives of a same concept, e.g., using or not the address, age or photo of a given class "Person". The IFM variability can express for instance the possible navigation alternatives to be activated or not. For instance on a shopping website, shortcuts providing quickly access to a given product can be configured. The CUI FM represents alternative representations of a widget: a panel (i.e., portion of UI displayed on a screen) can become a full window (i.e., displayed has full-screen) on mobile phones.
- *Mappings*: The variability of the mapping between IFM and Domain can be managed by a mapping FM. Interaction flow elements (UI states) can involve different concepts of the domain model.
- *Transformations*: Variability can be expressed also at the level of transformations. The variability that a transformation could convey (i.e., multiple output alternatives) can be expressed with FMs. The transformations impacted are (1) between IFM, Domain and CUI, (2) between CUI and ISM. The variability of (1) expresses the possible UI design choices: how an IFM state selection can be represented: a simple list, an indexed list, a tile list, etc. The variability in (2) depends on the target ISM and configures the final representation to be provided to end-users. For instance, a CUI simple list can be represented using, as output of the transformation, the following HTML markup alternatives: "<select>" or "<ul>".

By scoping the FM to a specific concern, our approach allows to focus only on the variations related to the underlying concern. In our approach, the different FMs enrich the existing UI design process accompanying, step-by-step, the design of models and their variability.

3.3.2 *Implementation: Model Transformation*

Before starting the whole transformation process (up to the first executable prototype), the various FMs that describe the possible configurations of the product line have to be aggregated. More details about this step is available in [43]. As a result, only one large FM is built allowing to produce, through model to model and model to text transformations the variants to be assessed. To aggregate the many FM, we can use the insert operator of Familiar [3] using the following expression see Listing 3.1.

Listing 3.1 Familiar insertion operator for building complete partial configuration

```
fml> insert MappingIFMDomain into CUITransformation.CUIList with mand
```

The implementation of our approach relies on an existing system that derives a UI from IFM and domain models using successive model transformations [41]. This initial system was not taking into account the configuration of the variability. The difference is that it uses the FM as a transformation configuration and generate a UI for each possible configurations depicted in the aggregate FM. We have built a small algorithm that produce a configuration model for each valid combination of the FM. Then this configuration model (conforms to a configuration metamodel) is used as an additional input model for the transformation. Nevertheless, we keep our tool initial behavior if no FM is specified: a default transformation is executed if it has no configuration (in ATL syntax: i.getConfiguration.oclIsUndefined()). The "getConfiguration" helper uses the explicit link between the input models (IFM/Domain) and the current configuration to be generated.

Listing 3.2 Excerpt of the default transformation used if no configuration is defined

```
rule selectionListViewDefault extends widgetEvents {
from
    i : SC!SelectionState(i.getConfiguration.oclIsUndefined())
to
    o : CUI!ListView (
    name <- i.name,
    id <- i.name.regexReplaceAll(' ',''),
    widgets <- i.domainElements->select(e| e.Type = #Image)->collect(e|
        ↪thisModule.image(e))
    )
}
```

We reused the rest of the transformation chain up to the application generation: CUI to ISM and ISM to Code. For each type of ISM (i.e., interaction State) a set of rules are produced corresponding to the possible variants. Each particular attribute of the widget (i.e., indexed and filtered) is also dependent on the configuration thus introducing additional conditional expressions (ListFilters and ListDividers conditions in Listing 3.3).

Listing 3.3 Excerpt of selection to tile list in CUI transformation including configuration helpers

```
helper context OclAny def: hasConfig(config:String) : Boolean =
if (self.getConfiguration.oclIsUndefined())
```

```
    then
        false
    else
        self.getConfiguration.WidgetName=config
endif;
...
rule selectionTileList extends widgetEvents {
from
    i : SC!SelectionState(i.hasConfig('TileList'))
using {
    conf:Configuration!Configuration = i.getConfiguration;
}
to
    o : CUI!TileList (
    name <- i.name,
    id <- i.name.regexReplaceAll(' ',''),
    icons <- i.domainElements->select(e| e.Type = #Image)->collect(e|thisModule
        ↪.image(e)),
    Listfilters <- if(conf.filtered) then filter else OclUndefined endif,
    listDivider <- if(conf.indexed) then divider else OclUndefined endif
    ),
    filter : CUI!Filter(
    filterRevealedList <- false),
    divider : CUI!Divider(
    autodivider <- true )
}
```

3.4 Evaluation of Configuration: Rapid Prototyping

End-user requirements are crucial in user centered design. They are often not formally defined: most of the time they are expressed as remarks on a portion of the produced or prototyped UI. Thus, in order to capture end-user requirements, UI designers have to propose various product versions (prototypes) to end-users. A common practice is to use rapid prototyping. Rapid prototyping is a user-centered iterative process where end-users give feedback on each produced prototype. Prototypes are usually mock-ups of UI drawn with dedicated tools (e.g., see balsamiq).[1] In order to show to the users an interaction experience closest to reality we should rely on higher fidelity and on living prototypes (i.e., the user should be able to interact with it). As a result, MDE provides us with semi-automatic generation capabilities. It allows a quick production of prototypes and many assessments in a limited amount of time. End-users will thus elicit the way they prefer interacting with the system, the best widgets and representations for their tasks. In fact, through these iterations they elicit the product configuration that best fits their needs.

In previous work [40, 42], we have tested the global usability of particular generated UI prototypes. We have also proposed a version that evaluate only a portion of the UIs produced by an SPL [43]. Configuration reconciliation can be a time consuming task, which may delay the product elicitation. Indeed the time for configuring and aligning all the partial configurations together can be a very consuming task even if we relax some FM constraints (i.e., getting an end-user feedback on generated UI after a one or two minutes configuration).

[1] http://balsamiq.com

Here we will try to evaluate the different UIs obtained by an SPL. The approach we have selected is an interactive genetic algorithm (IGA). It aims at reducing drastically the number of UI configurations to evaluate as well as the number of person necessary for this test. The idea is to converge quickly to good/acceptable UIs with a small number of algorithm iterations: limiting the effect of user fatigue. At each iteration the IGA will take into account a user ranking (e.g., like/dislike) of the proposed UI and propose, a mutated configuration for the next step. This will allow to test potentially many configurations whilst keeping only the convenient ones.

The approach we have followed here is decomposed in two steps. The first step consists in generating the set of UIs according to the possible configurations depicted in the FM. For the sake of performance (i.e., rapid evaluation between each IGA iteration) and because of interoperability issues we choose to produce first the set of evaluable UIs. Nevertheless the MDE-SPL generation mechanism could have been directly driven by the IGA produced configurations. The second step is iterative and consists in the evaluation of the produced UI variant and then production of next UIs to be evaluated thanks to the IGA.

3.4.1 Step 1: Deriving All Relevant Configurations

The idea is to generate all the possible configurations and then apply the UI generation process for each of them, as depicted in Sect. 3.3.2. Some constraints can also be added to existing ones [4]. They allow to remove product variants which are, predictably, not relevant for the task at hand. For instance, tile lists are not necessarily relevant without photos or images.

3.4.2 Step 2: Variant Assessment

We followed the recommendations of Nielsen regarding the minimal number of end users to involve in iterative user tests [30]. As such, we decided to run user tests with 5 participants in order to collect their feedback. For each iteration of the genetic algorithm, 10 UI configurations are suggested to testing (i.e., 2 per end-user). According to the predictions of the Poisson model [30], involving 5 users gives rise to an expected probability of reporting 85% of the usability problems for a simple UI. Given that usability problems impact their assessments, having at least 5 users should enable us to gracefully evolve to a reasonable good UIs at a reasonable cost regarding the number of involved end-users. The IGA is taking this user evaluation to propose, to all users, for the second round of evaluation a mutation of the best variants. A mutant is a new configuration that slightly different from the previous one, by e.g., changing one feature (add/remove).

The details of the implemented IGA is shown in Algorithm 1. First, the population is randomly initialized at line 1. After this, the evolution starts at line

Algorithm 1 Interactive genetic algorithm for data set creation in the contact list case study

```
    input: Genetic representation of a configuration = 20 bits, Population = 10 configurations,
    Users = 5
    output: Data set of user assessments
 1: population ← initializePopulation()
 2: while stopConditionNotSatisfied() do
 3:     foreach conf_i ∈ population do
 4:         conf_i.assignedUser ← assignUser(users,conf_i)
 5:     end foreach
 6:     foreach conf_i ∈ population, in parallel do
 7:         conf_i.fitness ← getUserFeedback(conf_i)
 8:         registerDataInstance(conf_i)
 9:     end foreach
10:     parents ← parentSelection(population)
11:     offspring ← crossover(parents)
12:     offspring ← repair(offspring)
13:     offspring ← mutate(offspring)
14:     offspring ← repair(offspring)
15:     population ← survivorSelection(offspring)
16: end while
```

2 until the stop condition is satisfied. In our case we used the termination condition when reaching a fixed number of generations in order to avoid user fatigue and time consumption. We can also limit the number of generations (iteration of the IGA): it limits the number of evaluations (against user-fatigue), but also stop before the evaluation curve is reverted (i.e., when producing mutant the quality of UI could regress because of user fatigue).

From line 3 to 5, each member of the population is assigned to one user for assessment. In our experimental settings, each of the 5 users is assigned to 2 members to cover the whole population. The user feedback for all the population is obtained from line 6 to 9. Once the fitness of the whole population is set, we can proceed to the parent selection for the next generation. The *parent selection operator* is based on a fitness proportionate selection (line 10). At line 11, the *crossover operator* is based on the half uniform crossover scheme [46]. The crossover, as well as the mutation operators of the GA, can end up with invalid configurations because of FM constraints. Existing works have solved this by penalizing the fitness function or trying to recover the configuration to a valid state. In our case, at line 12 and 14 we repair the offspring if hard constraints are violated. The *mutation operator* used at line 13 is uniform with $p = 0.1$. This mutation factor is meant to prevent a loss of motivation from users (i.e., user fatigue) by reducing the likelihood that they will keep assessing very similar UI configurations from the population, while thus enabling us to explore new regions of the configuration space. Finally, at line 15, the *survivor selection operator* is based on a complete replacement of the previous generation with the new generation.

3.5 Case Study

Contact Lists are widely used Human-Computer Interaction (HCI) applications to obtain personal information such as telephone numbers or email addresses. We can find them on mobile phones for personal use, communication systems for elderly people, corporate intranets or web sites. Despite of sharing the same objective, the final UI implementations are very diverse. We focus in this case study in building contact lists with corporate information.

User involvement is one of the key principles of the user-centered design method [15]. The literature includes a number of early works leveraging SPL techniques to deal with the management of HCI-specific variability using SPL-based approaches [32, 33]. In this case study dealing with UIs, we focus on early binding variability [26] where design alternatives are chosen at design-time. Therefore, this scenario does not focus on customization options which are performed by end-users themselves within the UI, nor to self-adaptive systems which corresponds to run-time binding of variability. The ISATINE framework [27] structures the adaptation life cycle into seven stages: goal specification (when the end-user defines her goals for adaptation), adaptation initiative (who is taking the responsibility to trigger the adaptation), adaptation specification (what could be performed to ensure an adequate adaptation), adaptation application (performed by the system itself), adaptation transition (how to convey the transition between the stage before adaptation and after), adaptation interpretation (how to interpret the results of the adaptation), and adaptation evaluation (how to evaluate the results of the adaptation with respect to the initial goals). If these seven stages are considered, the adaptation specification and application are particularly ensured by our approach. The transition should be ensured by other techniques.

3.5.1 The Contact List Example

Figure 3.4 presents the FM defining the variability of the Contact List application domain. The FM was created by HCI experts from the Luxembourg Institute of Science and Technology (LIST) with whom we collaborated in this case study. This FM encodes knowledge about the interface design defining a configuration space of 1365 valid configurations. UI design choices, even for this apparently simple case, give raise to voluminous configuration spaces.

The `ContactList` variability is decomposed into three main features: `List` depicting the possible choices to be made in terms of widgets for representing the list, `Master Detail Interface` which states the global layout of the application and `Details Grid` which sets the layout for the detailed information of a person. The `ListType` variability defines the different alternatives of List widgets: `DropDownList` is a select box showing only one item when inactive, `ListView` is a classic navigation list and `TileList` is a list of thumbnails represented as tiles. The `Indexed` optional feature separates and ranks the list

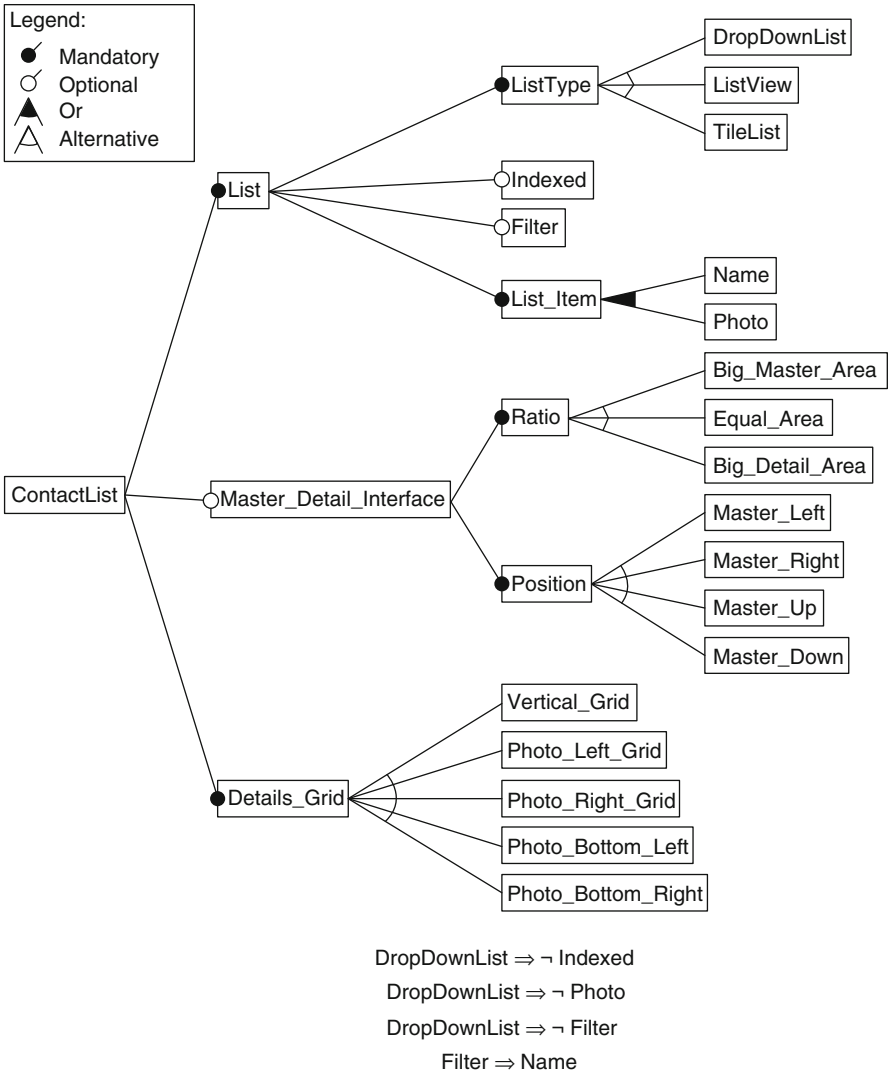


Fig. 3.4 Contact list feature model

items by the first letter of the name. The `Filter` optional feature adds a search functionality implemented through a text box that automatically filters the list items according to the text introduced by the user. The `ListItem` consists of the `Name` of the person or the `Photo`, or both. Four cross-tree constraints are shown in the feature diagram which are related to these features. Concretely, the `DropDownList` feature excludes `Indexed`, `Photo` and `Filter` features. Also, the `Filter` feature requires the `Name` in the `ListItem` feature.

The `Master Detail Interface` is an optional feature that split the screen in two parts: the master and the detail. The master contains the list while the details interface, after a selection in the master, shows the corresponding contact information. There is variability concerning the `Ratio` of the screen split and the `Position` of the master interface. Finally, the `Details Grid` feature represents different alternatives to organize the contact information on the screen (e.g., telephone number, address etc.) as for example including all information in one column or determine the position of the textual information with respect to the photo.

The SPL has been implemented using the Variability-aware Model-Driven UI design framework [44] based on AME (Adaptive Modeling Environment) [20] which is able to derive, through source code generation, any configuration of the presented FM. The target framework for the derived products is the JQueryMobile web framework.[2]

Screenshots demonstrate the diversity of UIs that can be obtained (they have been anonymized to avoid displaying personal information): Figs. 3.5 and 3.6 present UI variants from which we enumerate their corresponding features. Figure 3.5 shows an example that includes `ListView` with only the `Name` (see left side of the figure). The list view is neither `Indexed` nor has a `Filter` feature. It has `Master Detail Interface` with `Equal Area` and `Master Left` given that the screen is split in two identical parts with the list (master) at the left and the details grid at the right. The details are displayed with `Photo Right Grid`.

Figure 3.6a, b show how the `TileList` is realized (see right side of the figure) and Fig. 3.6c how the `DropDownList` is displayed (see left side of the figure). Figure 3.6d shows a UI variant whose configuration does not have a master detail. It only displays on the screen either master or detail (note the presence of the back button at the top left of the screenshot for coming back to the master). In the case of master detail, the ratio indicates whether we have a big master interface with a small details interface (e.g., Fig. 3.6c) or vice-versa. Alternatively, we can have the split into two equal parts (Figs. 3.5 and 3.6a).

Fig. 3.5 Configuration and screenshot of its associated contact list UI variant

[2]JQueryMobile web framework: https://jquerymobile.com

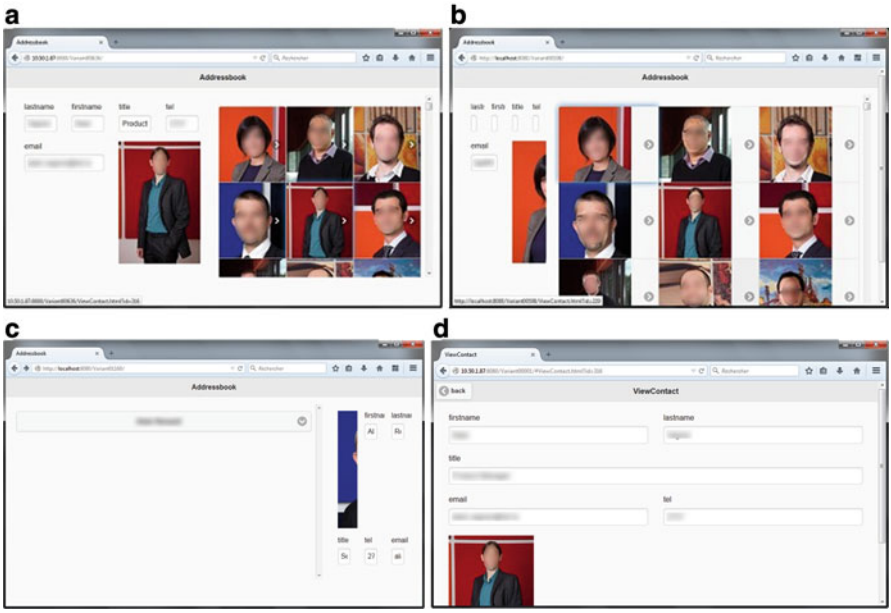


Fig. 3.6 Screenshots of derived variants from the contact list SPL and enumeration of their associated features. (**a**) Tile List, Photo, Master Detail (with Equal Area and Master Right) and Details Grid with Photo Bottom Right. (**b**) Tile List, Photo, Master Detail (with Big Master Area, Master Right). (**c**) DropDownList, Name, Master Detail (with Big Master Area, Master Left). (**d**) Master Detail Feature (The list variability is not illustrated in this figure) and Details Grid with Photo Bottom Right

The `Position` variability is related to a horizontal or vertical split of the screen and whether the master is in one side or the other (in Fig. 3.6b the master and detail have been swapped). If the `Master Detail Interface` feature is not selected in a configuration, the window split will be replaced during the navigation: one first window for selecting the person to be displayed and the other one for seeing the details. Finally, the `Details Grid` feature represents different alternatives to organize the contact information on the screen. For instance, in Fig. 3.6a the grid is four columns and two rows whereas in Fig. 3.6d it is two columns and four rows.

3.5.2 Defining Configurations Chromosome

One important decision to implement the genetic algorithm is how to represent the individuals (the configurations in our case). We consider a binary array. Figure 3.7 shows an example of the chromosome of an individual that conforms to the Contact List SPL. The phenotype consists of the non-abstract features of the FM. Concretely, the leaves of the FM and the `Master Detail Interface` feature (see Fig. 3.4) are coded on a binary string of 20 bits. The features are the fixed indexes of the array

Fig. 3.7 Example chromosome of an individual of the contact list SPL

DropDownList	ListView	TileList	Indexed	Filter	Name	Photo	MasterDetailInterface	BigMasterArea	EqualArea	BigDetailArea	MasterLeft	MasterRight	MasterUp	MasterDown	VerticalGrid	PhotoLeftGrid	PhotoRightGrid	PhotoBottomLeft	PhotoBottomRight
0	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	0	1	0	0

where the value 1 means that the feature is activated and 0 that it is not. Representing a FM configuration chromosome as an array of bits is a common practice in the use of genetic algorithms in SPLE [17, 24]. As presented before, the details of the implemented IGA are shown in Algorithm 1.

3.6 Case Study: Experimentation

The objectives in this case study is to investigate whether:

1. The variant assessment selects better configurations than a randomized algorithm for a given number of iterations. To that end:
   - We quantitatively evaluated the improvement of the user scores through the IGA compared to random selection within the configuration space.
   - We study the diversity of the population along the generations to show the convergence of the IGA towards relevant UI designs.
2. The best UIs are configurations that usability experts confirm as relevant UI designs. We qualitatively discuss the findings with a usability expert and we checked if the top ranked configurations are close to configurations elicited by usability experts.

We conducted two independent experiments in two organizations: LIST and SnT. Figure 3.8 presents the results of the IGA for the two organizations. On the horizontal axis we have the different generations and the vertical axis is the mean of the scores of the user assessments for this generation. We show the score mean along the six generations including the standard deviations. An ascendant progression means that for each new generation, globally, the UI variants are being better appreciated by the pool of users. In Fig. 3.8a we observe a quick ascension until generation four while in Fig. 3.8a we observe the ascendant progression starting at generation two.

Despite that we do not have the explanation for the descending effect in LIST case for generations five and six, we consider that it is caused by the experience

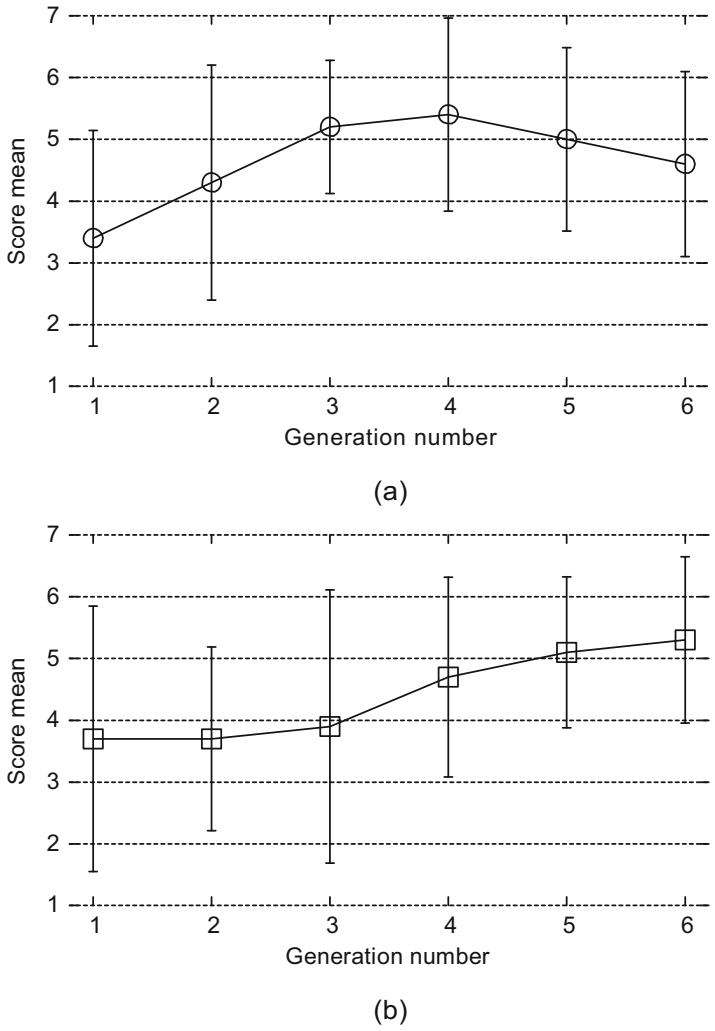


Fig. 3.8 Results of the genetic algorithm evolution in the two organizations. (**a**) LIST. (**b**) SnT

that the users had in UI design. The score mean improved quickly from generation one to four by filtering really inappropriate variants, and then decreased a little bit because of their capacity to criticize the proposed variants. They may not evaluate the variant itself but its capability to be different from what they already evaluated. Furthermore, some of these critics were related to non-variability related issues. Other possible explanation could be user fatigue.

In order to observe whether the IGA tends to converge, Fig. 3.9 shows the progression of the generations in the two dimensional space of mean score and diversity. We calculated the genotype diversity along the different generations (*g1* to *g6*) as the average of the Hamming distance of all pair of configurations in the

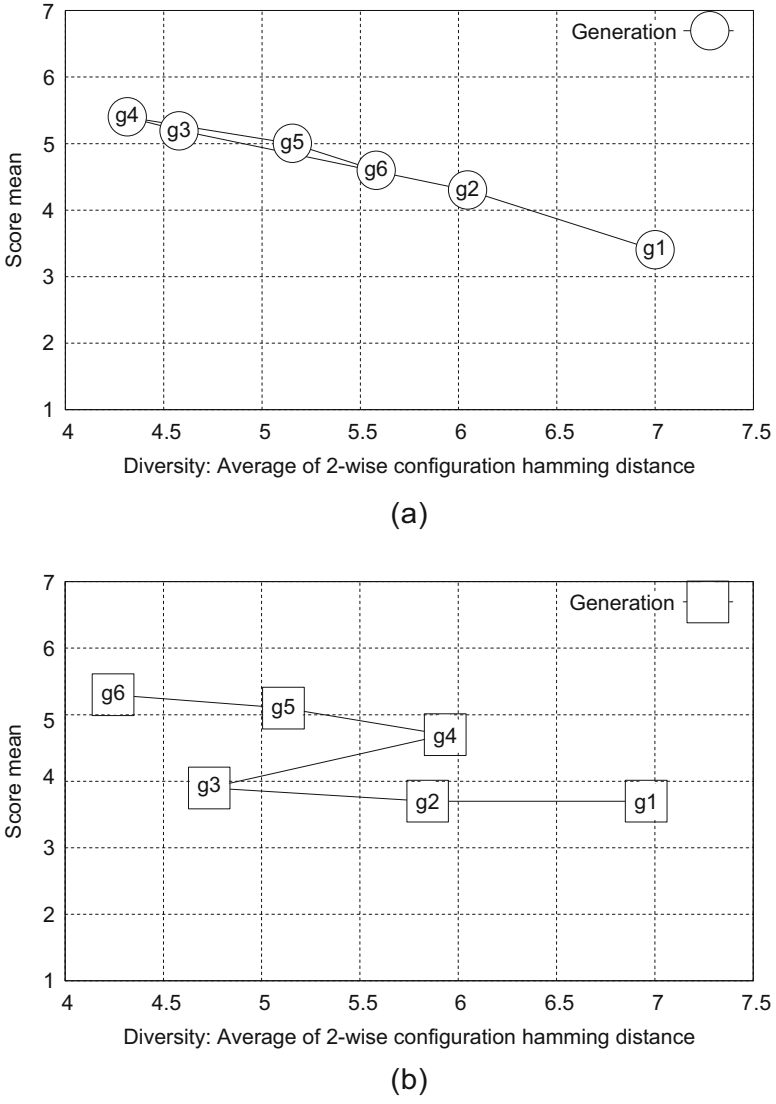


Fig. 3.9 Generations progress in terms of mean score and diversity. (**a**) LIST. (**b**) SnT

generation. The diversity decreases if we approach to the left side of the horizontal axis. For example, we can observe how the diversity is not increasing more than its value at *g1* which is the randomly created population. For LIST, as shown in Fig. 3.9a, *g4* has both the lowest diversity and the maximum mean score. In the case of SnT, as shown in Fig. 3.9b, the last generation (*g6*) has both the lowest diversity and the maximum mean score. In the LIST case, the user pool was able to reach better variants for them earlier.

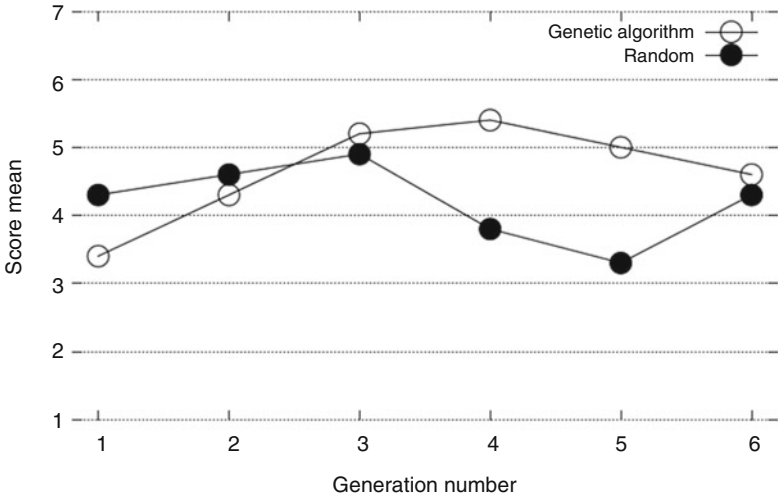
UI Quality Improvement. Regarding the first hypothesis, we evaluate if our process based on evolutionary techniques selects better variants than a randomized algorithm for a given number of iterations. We repeated the experiments with the same participants using random selection. In this approach, for each generation, 10 configurations were automatically selected from the viable space which is the same size of the population that we used for the IGA. Basically, for the random selection, we used the same operator as the one used for seeding in the IGA. Despite that we still call each group of 10 random configurations a generation, no genetic information was propagated from one generation to the next. Figure 3.10 shows the results of the genetic algorithm and the random selection in order to compare them. The most important observation is that random selection failed to obtain a global score mean greater than 5 in any of the generations while the genetic algorithm did achieve it. We can see how the genetic algorithm outperforms the random selection approach except in the first two generations where the effect of evolution is still trying to find relevant regions of the configuration space.

Table 3.1 presents the representative improvements obtained in the two independent experiments by comparing the global score mean. The global score mean is the mean of the assessment scores in all the generations. The genetic algorithm approach has a global score mean which is around 0.5 points better (i.e., 0.45 in LIST and 0.55 points in SnT).

The proposed genetic algorithm produced better results over the generations than the random approach. The algorithm tends to converge to a solution in this search of better UI configurations. To prove this, we computed the diversity of the members of each generation. If the diversity has a tendency to decrease, it is a sign of convergence. Figure 3.11 shows the graph of the results at organization LIST and SnT for both the genetic algorithm and the random process. We can see how the random approaches in both organizations do not decrease the diversity while, for these 6 generations, we observe how the genetic algorithm performs better than the random approach to reduce the diversity. The random approach failed to decrease the diversity to values lower than 5 while this was achieved by the genetic algorithm approach. As result, we can conclude that, compared to the random approach, we both increase the global mean score and we reduce the diversity along the generations. These two aspects allow the genetic algorithm to try to converge to optimal or sub-optimal solutions which means to relevant UI designs.

Usability Expert Analysis. In order to confirm our second hypothesis we required a usability expert with nine years of experience to assess that the better variants found by our approach satisfy usability criteria. This expert is independent in order to provide an impartial assessment. He does not belong to the team that developed the considered project, nor participated during the variant assessment. We summarize the expert qualitative evaluations on the three relevant variants shown in Fig. 3.12:

– The first variant, shown in Fig. 3.12a, is the simplest list with no master detail. It addresses several usability criteria [37] such as low workload, explicit control,



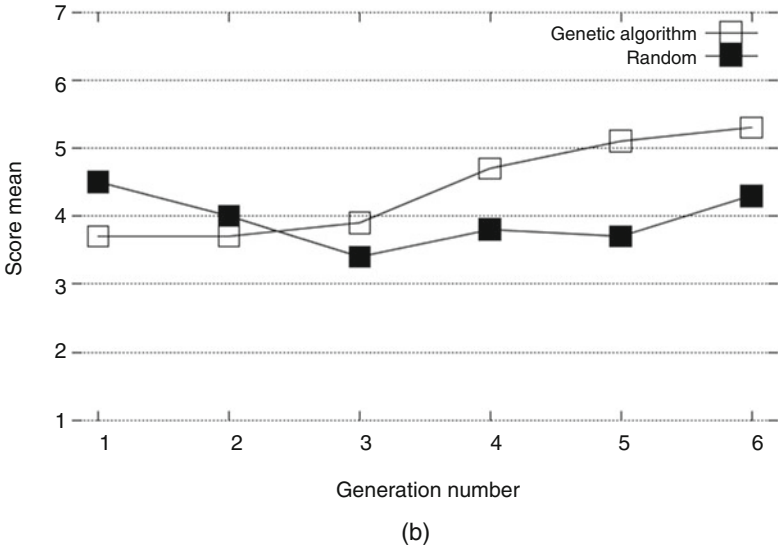


Fig. 3.10 Comparing variant selection based on genetic algorithm and random. (**a**) LIST. (**b**) SnT

Table 3.1 Global score mean evaluation

	GA	Random
LIST	4.65	4.20
SnT	4.40	3.95

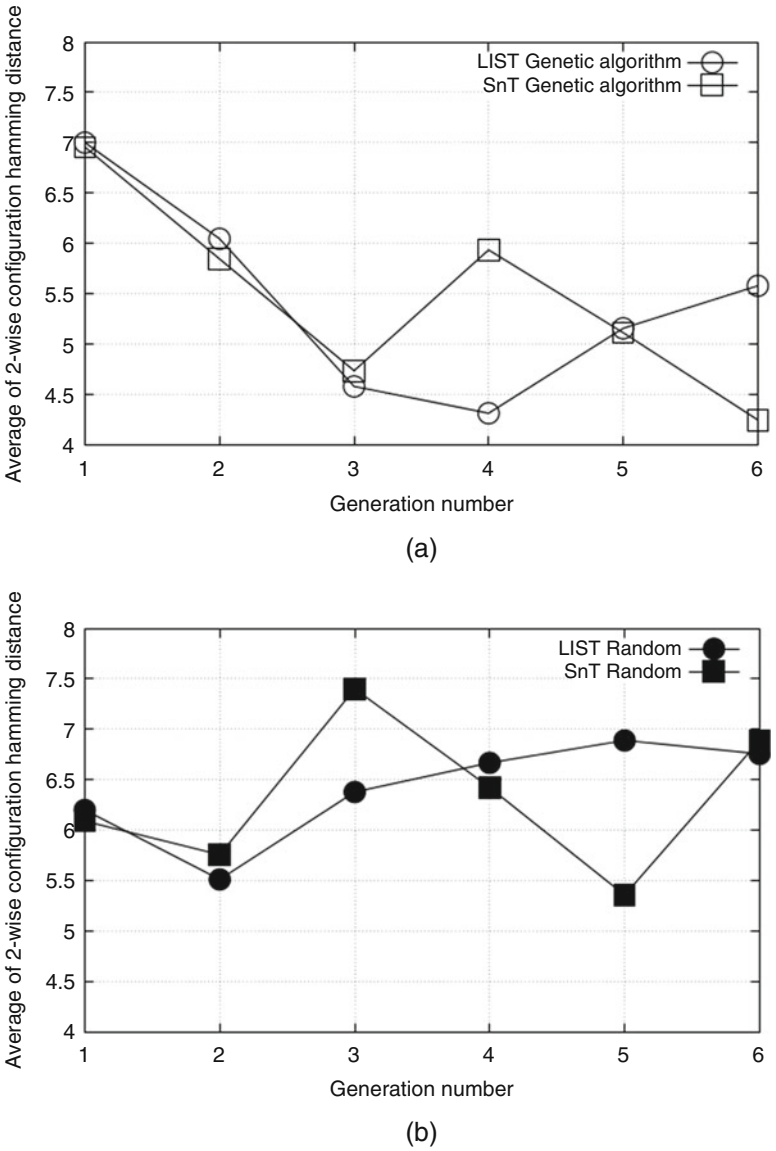


Fig. 3.11 Genotype population diversity. (**a**) Genetic algorithm. (**b**) Random

homogeneity/consistency or compatibility with traditional contact applications. The search bar and the simplicity of the UI allows the end user to go directly to what he/she is looking for. However, as drawback, it is not possible to browse through the contacts' photos or to do a visual research if the name of the person is unknown.

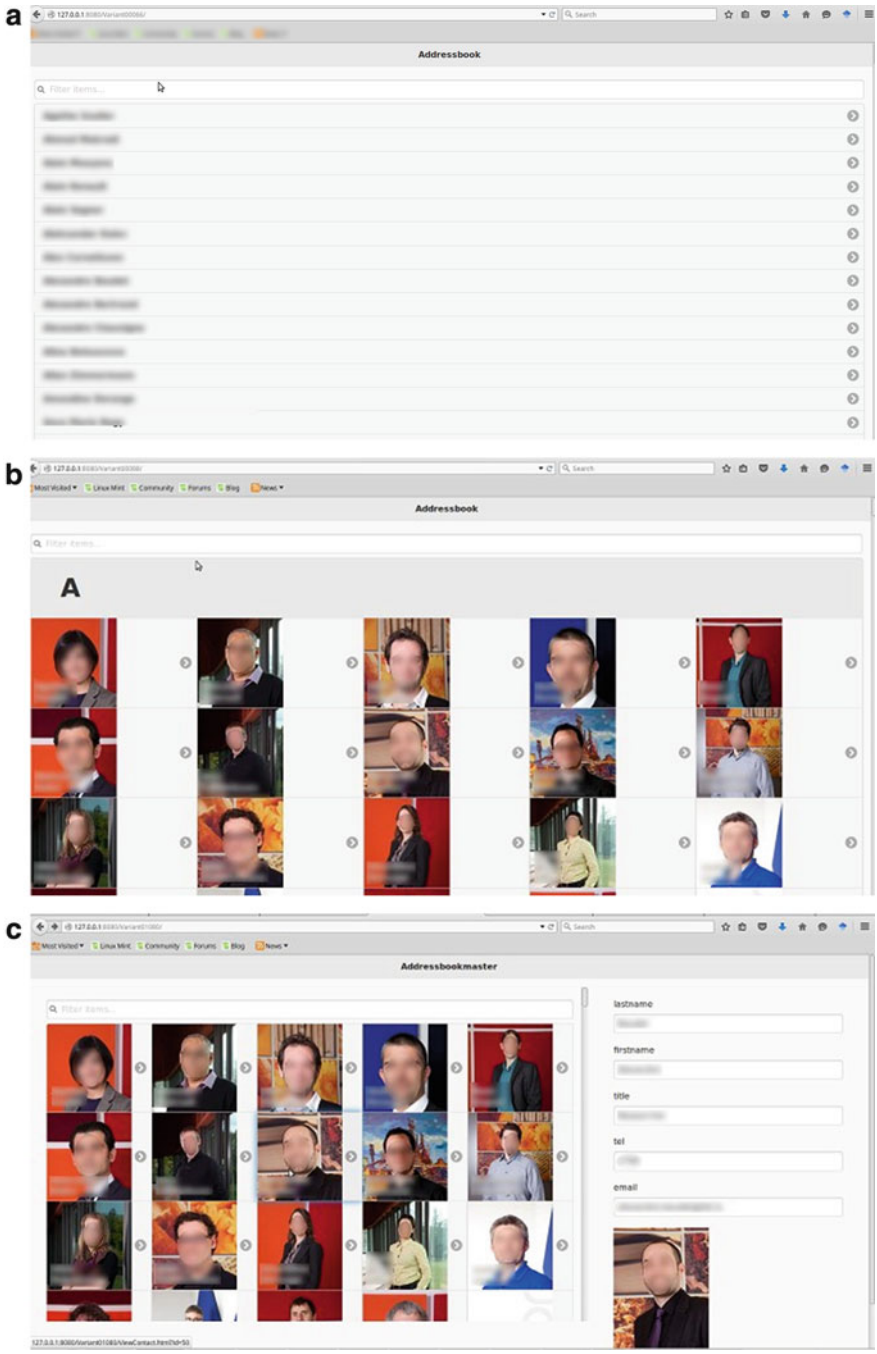


Fig. 3.12 Screenshots of relevant variants. (**a**) ListView, Name and no MasterDetailInterface. (**b**) TileList, Name, Photo, Indexed, Filter and no MasterDetailInterface. (**c**) TileList, Name, Photo, Filter, MasterDetailInterface (with Big Master Area and Master Left) and Vertical Grid

– The second variant, shown in Fig. 3.12b, has a better appearance (aesthetic consideration) and has more information (i.e., photo and index). It also complies well with Scapin and Bastien's usability criteria [37]. Notably the adaptability criteria is well implemented here: the application can be convenient to the different situations of use (e.g., on large screen and small screen display, etc.). However, it seems visually overloaded. Reducing the number of persons displayed in the list can be an option. Another important point noticed is that the users can just play with the UI (e.g., browse through colleague photos) and be distracted from the prescribed task.
– The third variant, reproduced in Fig. 3.12c, is very close to the previous one, except for the master/detail pattern. It also complies with most of the usability criteria. The list of persons is more compact than the previous variant (Fig. 3.12b) giving a better impression. The information is accessible directly without the need to navigate which is a plus for large screens but not necessarily the best solution. In the configuration with a Master Detail interface, the layout is important, and in this variant the vertical grid fits perfectly with this layout.

The usability expert claimed that the relevant variants that have emerged from applying the approach satisfy most of the usability criteria.

3.7 Conclusion and Perspectives

In this article, we addressed the issue related to UI variability. UI variability holds numerous facets (e.g., graphical design, development, usability, etc.) due to the diversity of stakeholder profiles whom may contribute to the UI development life cycle, such as, but not limited to, end-users. Moreover, in UI design, one encounters frequently the difficulty to align the products with fuzzily defined user requirements. This complexity can lead to an inefficient UI design process, which has an impact on the UI design costs.

Therefore, we proposed an approach to manage UI variability based on MDE and SPL, integrating SPL management into our current MDE UI design process. In order to build a viable product, the stakeholders have to confront their viewpoints when configuring products. This is the general approach we have illustrated here, adopted for rapid prototyping.

This article focused on the end-users as peculiar stakeholders of the design process. Such stakeholders can intervene in some specific parts of the process using partial FM: i.e., help in choosing some features. Nevertheless the potential indirection between feature and the resulting UI could make things unclear. This is why we should still rely on user assessments.

When using SPL, we can obtain many variants of the same UI, thus making it difficult to assess for end-users. How end-users would assess more than 800 variants? Even, if they had time to evaluate it, when assessing many similar UIs the user-fatigue will provide biased results. We could have used partial configuration to split the problem and focus on a specific element such as in [43]. However, it will not have provided results for the overall interaction experience.

We have experimented with an Interactive Genetic Algorithm that helps to assess and deal with many variants produced by an SPL approach. It helps to reduce the number of UI to be tested by end-users and to find consensus on a relevant versions (i.e., of a good quality). First, it facilitates the exploration of the design space (as defined in the FM) and with a rather small portion of the possible configurations. Second, it helps to assess the variants with a group of users reducing some personal subjectivity.

One future direction of this work would be to propose a ranking of the features which influence user decisions the most. As such we would be able to predict the configurations which are not relevant or the most adapted to specific interaction situations.

Acknowledgements This work has been partially supported by the FNR CORE Project MoDEL C12/IS/3977071. The work of Jabier Martinez is funded by the AFR grant agreement 7898764. The work of Alfonso García Frey is partially co-funded by Yotako S.A. and the FNR Luxembourg under the AFR grant agreement 7859308. Special thanks to Alain Vagner for his contributions.

References

1. Abrahão S, Iborra E, Vanderdonckt J. Usability evaluation of user interfaces generated with a model-driven architecture tool. In: Maturing usability. London: Springer; 2008. p. 3–32.
2. Acher M, Collet P, Lahire P, France RB. Separation of concerns in feature modeling: support and applications. In: Proceedings of the 11th Conference on Aspect-Oriented Software Development. 2012.
3. Acher M, Collet P, Lahire P, France RB. Familiar: a domain-specific language for large scale management of feature models. Sci Comput Program. 2013;78(6):657–81.
4. Barreiros J, Moreira A. Soft constraints in feature models. In: Proceedings of ICSEA 2011: The Sixth International Conference on Software Engineering Advances. IARIA XPS Press; 2011, p. 136–141. ISBN: 978-1-61208-165-6.
5. Batory D, Azanza M, Saraiva J. The objects and arrows of computational design. In: Model driven engineering languages and systems. Berlin: Springer; 2008. p. 1–20.
6. Benavides D, Martín-Arroyo PT, Cortés AR. Automated reasoning on feature models. In: CAiSE. 2005. p. 491–503.
7. Benavides D, Segura S, Ruiz-Cortés A. Automated analysis of feature models 20 years later: a literature review. Inf Syst. 2010;35(6):615–36.
8. Brummermann H, Keunecke M, Schmid K. Variability issues in the evolution of information system ecosystems. In: Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems. 2011.
9. Bühne S, Lauenroth K, Pohl K. Why is it not sufficient to model requirements variability with feature models. In: Workshop on Automotive Requirements Engineering (AURE04), at RE04. 2004.
10. Calvary G, Coutaz J, Thevenin D, Limbourg Q, Bouillon L, Vanderdonckt J. A unifying reference framework for multi-target user interfaces. Interact Comput. 2003;15(3):289–308.
11. do Carmo Machado I, McGregor JD, de Almeida ES. Strategies for testing products in software product lines. ACM SIGSOFT Softw Eng Notes. 2012;37(6):1–8.
12. Clements P, Northrop L. Software product lines. Boston/London: Addison-Wesley Boston; 2002.
13. Czarnecki K, Antkiewicz M, Kim CHP, Lau S, Pietroszek K. Model-driven software product lines. In: Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications. ACM; 2005. p. 126–27.

14. Czarnecki K, Helsen S, Eisenecker U. Staged configuration through specialization and multilevel configuration of feature models. Softw Process Improv Practice. 2005;10(2): 143–69.
15. DIS I. 9241-210: 2010. Ergonomics of human system interaction-part 210: human-centred design for interactive systems. Geneva: International Standardization Organization (ISO); 2009.
16. Eiben AE, Smith JE. Introduction to evolutionary computing. Berlin/London: Springer; 2003.
17. Ensan F, Bagheri E, Gašević D. Evolutionary search-based test generation for software product line feature models. In: Ralyté J, Franch X, Brinkkemper S, Wrycza S, editors. Advanced information systems engineering. Lecture notes in computer science, vol. 7328. Berlin/Heidelberg: Springer; 2012. p. 613–28.
18. Gabillon Y, Biri N, Otjacques B. Designing multi-context UIs by software product line approach. In: ICHCI'13. 2013.
19. García JG, Vanderdonckt J, González-Calleros JM. Flowixml: a step towards designing workflow management systems. Int J Web Eng Technol. 2008;4(2):163–82. http://dx.doi.org/10.1504/IJWET.2008.018096
20. García Frey A, Sottet JS, Vagner A. Ame: an adaptive modelling environment as a collaborative modelling tool. In: Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems. New York: ACM; 2014. p. 189–92.
21. García Frey A, Sottet JS, Vagner A. Towards a multi-stakehoder engineering approach with adaptive modelling environments. In: Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems. New York: ACM; 2014. p. 33–8.
22. García Frey A, Sottet JS, Vagner A. A multi-viewpoint approach to support collaborative user interface generation. In: 19th IEEE International Conference on Computer Supported Cooperative Work in Design CSCWD. 2015.
23. Henard C, Papadakis M, Perrouin G, Klein J, Traon YL. Multi-objective test generation for software product lines. In: SPLC. 2013. p. 62–71.
24. Henard C, Papadakis M, Perrouin G, Klein J, Heymans P, Traon YL. Bypassing the combinatorial explosion: using similarity to generate and prioritize t-wise test configurations for software product lines. IEEE Trans Softw Eng. 2014;40(7):650–70.
25. Johansen MF, Haugen Ø, Fleurey F, Eldegard AG, Syversen T. Generating better partial covering arrays by modeling weights on sub-product lines. In: MoDELS. 2012. p. 269–84.
26. Kang KC, Lee J, Donohoe P. Feature-oriented project line engineering. IEEE Softw. 2002;19(4):58–65.
27. López-Jaquero V, Vanderdonckt J, Simarro FM, González P. Towards an extended model of user interface adaptation: the isatine framework. In: Gulliksen J, Harning MB, Palanque PA, van der Veer GC, Wesson J, editors. Engineering Interactive Systems – EIS 2007 Joint Working Conferences, EHCI 2007, DSV-IS 2007, HCSE 2007, Salamanca, Mar 22–24, 2007. Selected Papers. Lecture notes in computer science, vol. 4940. Springer; 2007. p. 374–92. http://dx.doi.org/10.1007/978-3-540-92698-6_23
28. Mannion M, Savolainen J, Asikainen T. Viewpoint-oriented variability modeling. In: COMPSAC'09. 2009.
29. Martinez J, Lopez C, Ulacia E, del Hierro M. Towards a model-driven product line for web systems. In: 5th Model-Driven Web Engineering Workshop MDWE. 2009.
30. Nielsen J, Landauer TK. A mathematical model of the finding of usability problems. In: Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems. ACM; 1993. p. 206–13.
31. OMG. IFML – interaction flow modeling language. 2013.
32. Pleuss A, Botterweck G, Dhungana D. Integrating automated product derivation and individual user interface design. VaMoS. 2010;10:69–76.
33. Pleuss A, Hauptmann B, Dhungana D, Botterweck G. User interface engineering for software product lines: the dilemma between automation and usability. In: EICS. New York: ACM; 2012. p. 25–34.

34. Pohl K, Böckle G, Van Der Linden F. Software product line engineering: foundations, principles, and techniques. Berlin: Springer; 2005.
35. Rosenmüller M, Siegmund N. Automating the configuration of multi software product lines. In: VaMoS. 2010. p. 123–30.
36. Sayyad AS, Menzies T, Ammar H. On the value of user preferences in search-based software engineering: a case study in software product lines. In: ICSE. 2013. p. 492–501.
37. Scapin DL, Bastien JC. Ergonomic criteria for evaluating the ergonomic quality of interactive systems. Behav Inf Technol. 1997;16(4–5):220–31.
38. Schlee M, Vanderdonckt J. Generative programming of graphical user interfaces. In: Proceedings of the Working Conference on Advanced Visual Interfaces, AVI'04. New York: ACM; 2004. p. 403–6. http://doi.acm.org/10.1145/989863.989936
39. Shneiderman B. Promoting universal usability with multi-layer interface design. In: Proceedings of the 2003 Conference on Universal Usability, CUU'03. New York: ACM; 2003. p. 1–8. http://doi.acm.org/10.1145/957205.957206
40. Sottet JS, Vagner A. Genius: generating usable user interfaces. arXiv preprint arXiv:1310.1758; 2013.
41. Sottet JS, Vagner A. Defining domain specific transformations in human-computer interfaces development. In: 2nd Conference on Model-Driven Engineering for Software Developement. 2014.
42. Sottet JS, Calvary G, Coutaz J, Favre JM. A model-driven engineering approach for the usability of plastic user interfaces. In: Engineering Interactive Systems. Berlin/New York: Springer; 2008. p. 140–57.
43. Sottet JS, Vagner A, García Frey A. Model transformation configuration and variability management for user interface design. In: International Conference on Model-Driven Engineering and Software Development. Springer International Publishing; 2015. p. 390–404.
44. Sottet JS, Vagner A, García Frey A. Variability management supporting the model-driven design of user interfaces. In: Modelsward. 2015.
45. Sumner T, Davies S, Lemke AC, Polson PG. Iterative design of a voice dialog design environment. In: Wixon DR, editor. Posters and Short Talks of the 1992 SIGCHI Conference on Human Factors in Computing Systems, CHI 1992, Monterey, 3–7 May 1992. New York: ACM; 1992. p. 31. http://doi.acm.org/10.1145/1125021.1125050.
46. Syswerda G. Uniform crossover in genetic algorithms. In: Schaffer JD, editor. Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA). Morgan Kaufmann; 1989, p. 2–9. ISBN: 1-55860-066-3.
47. Takagi H. Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation. Proc IEEE. 2001;89(9):1275–96.
48. White J, Dougherty B, Schmidt DC, Benavides D. Automated reasoning for multi-step feature model configuration problems. In: Proceedings of the 13th International Software Product Line Conference. 2009.