"Adapt-First: a MDE Transformation Approach for Supporting User Interface Adaptation"

Khaddam, Iyad ; Mezhoudi, Nesrine ; Vanderdonckt, Jean

Abstract

Adapting user interfaces to different contexts of use is essential to enhance usability. Adaptation enhances user satisfaction by meeting changing context of use requirements. However, given the variety of contexts of use, and the significant amount of involved information and contextual treatments, transformations of user interface models that consider adaptation become complex. This complexity becomes a challenge when trying to add new adaptation rules or modify the transformation. In this paper, we present “Adapt-first”, an adaptation approach intended to simplify adaptation within model based user interfaces. It capitalizes on differentiating adaptations and concretization via two transformation techniques: concretization and translation. First-Adapt approach aims at reducing complexity and maintenance efforts of transformations from a model to another.

Document type : *Communication à un colloque (Conference Paper)*

Adapt-First: a MDE transformation approach for supporting User Interface Adaptation

Iyad Khaddam, Nesrine Mezhoud, Jean Vanderdonckt
Louvain Interaction Laboratory, Louvain School of Management (LSM) – Place des Doyens, 1
Université catholique de Louvain (UCL) – B-1348 Louvain-la-Neuve, Belgium
{iyad.khaddam, nesrine.mezhoudi, jean.vanderdonckt}@uclouvain.be

***Abstract*—Adapting user interfaces to different contexts of use is essential to enhance usability. Adaptation enhances user satisfaction by meeting changing context of use requirements. However, given the variety of contexts of use, and the significant amount of involved information and contextual treatments, transformations of user interface models that consider adaptation become complex. This complexity becomes a challenge when trying to add new adaptation rules or modify the transformation. In this paper, we present "Adapt-first", an adaptation approach intended to simplify adaptation within model based user interfaces. It capitalizes on differentiating adaptations and concretization via two transformation techniques: Concretization and translation. First-Adapt approach aims at reducing complexity and maintenance efforts of transformations from a model to another.**

Keywords— User Interfaces, Adaptation, Model Driven Engineering, Transformation, Reification

I. INTRODUCTION

Model-based user interface (MB-UI) development approaches gained a lot of attention due to their potential benefits during the past decades. Such assets are mainly inherited from model-based development and Model-Driven Engineering (MDE) [1,2]. One of the main potential benefits is the support of exploring alternative designs which allow the production of different designs for different contexts of use while maintaining consistency [3].

On the other hand, in the expanding world of communications and technologies, UI context-awareness becomes a necessity. Adapting the interface in the context of user enhances the user's experience and improves the UI quality and usability [4,5]. Cameleon Reference Framework (CRF) defines the context of use as the triplet: < user, platform, and environment> [6].

MBUI development process is based mainly on Models and transformations. Where models denote abstract specifications of the interface and transformation present the mappings relationship inter and intra abstraction levels. An extensive description of potential transformations in CRF are presented in [7]. Fours transformations are identified: concretization, abstraction, translation, and reflexion.

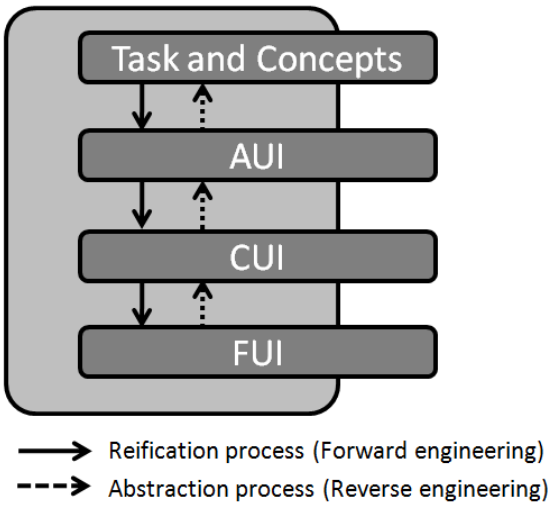


Figure 1. The Cameleon Reference Framework.

- Concretization transforms a particular model into another one of a lower level of abstraction, until executable/interpretable code is reached.
- Abstraction is an operation that transforms a UI representation from any level of abstraction to a higher level. Reverse engineering of user interfaces is a typical example of abstraction.
- Translation is an operation that transforms a description intended for a particular context of use into a description at the same level of abstraction.
- Reflexion is an operation that transforms a model into another one at the same level of abstraction for the same context of use (as opposed to different contexts of use as for translation).

The literature convoys several examples on adaptive model-driven UIs. Many works in MB-UI refer to the Cameleon Reference Framework (CRF). Several have been applied over the user interfaces engineering process, in compliance with different UI abstraction levels and the CRF. These researches focus on the definition of languages covering different abstraction levels describing the UI, and also the correspondences between levels (mappings) and

transformations. An overview and evaluation of these systems is reported by [8].

Models can be exploited at runtime to recognize context changes and support on the fly adaptation [1]. Model based UI are intended to reproduce all potential benefits of model-based development and Model-Driven Engineering (MDE) in general. Almost all benefits and shortcoming of Model-Based User Interfaces are reported by W3C in [1]. We would refer to the following advantages:

- To reduce the gap between requirements and implementation: Since models are able to define precisely functional requirement to better match user expectations.
- To improve the communication between stakeholders by explicit models, this explicitness enhances mainly the use (understanding, perceiving, comparing…) of models and define semantics of each models.
- To enhance the productivity via the generation of code, the possibility of reuse and reducing errors…
- To consider context evolution at runtime and enable on the fly adaptation.

Still, building the perfect model for UIs looks far beyond reach. Adding flexibility to modeling and supporting customization of the UI is vital to the success of MDE approaches. Support for context-awareness, flexibility and customization is usually implemented in the reification transformation, which results in increasing complexity of the reification. This increased complexity leads to difficulties in incorporating later modifications and updates on the transformation engine which affects the ability to improve adaptation with newly acquired established rules on adaptation to contexts of use.

We believe that simplifying the reification transformation improves the ability of the UI to support new context rules. Improvements include reducing the complexity and the maintenance efforts to accommodate new contexts of use. Accordingly, we propose an approach, Adapt-First, that aims at simplifying the reification process.

Our approach considers the adaptation as a translation (model-to-self transformation), while the concretization is the transformation from an abstract model to a lower one. Thus, the reification process passes through two steps: (1) the adaptation (translation): to adapt the UI to the context of use, and (2) the concretization: transform the model to a more concrete one. An intermediate model is employed to enable the passage from one step to the other.

The remainder of this paper is structured as follows. The following section refers to related works in the MB-UI domain. The third section presents the adapt-first approach, the method to apply the approach and some illustrative examples on how to apply the method on two adaptation cases at two different levels of abstraction. Section four shows a case study and implementation. We give an overview on the tools used and the transformation tool design (the translation and the concretization) and illustrate the implementation of three adaptation rules to culture and to platform. We conclude and explain future works in section five.

II. Related Works

Many UI models were developed since the 1980's, Researches in [9,10] propose model-based approaches for developing and adapting UIs at design-time and run-time. Such approaches generally use techniques of model-driven engineering (MDE) for the automatic generation of the UI. Addressing issues concerning the simplification of the process of user interface generation, and providing an infrastructure to allow applications to run in different contextual circumstances with different capabilities is a common purpose in [11]. To that end, different approaches have been proposed addressing adaptation problems; many of them stimulate adaptation via an adaptive behavior [12, 13, 14, 9]. Generally, for model based approaches, the stepwise development life cycle put forward a separation of concerns providing a good basis for producing a well-structured system, besides facilitating implementation itself as well as maintenance. Several adaptations have been applied over the UI engineering process, in compliance with diverse abstraction levels and the CRF. Investigations were focused on the definition of languages covering different abstraction levels describing the UI, rather than correspondences between levels (mappings) and transformation functions. Usixml is an example [15].

In this paper we focus on model transformation. Quite few studies addressed transformation from different perspectives. An extensive domain analysis of existing transformation techniques is presented by [Krzysztof]. The study result in a taxonomy-based classification of the existing model-to-model transformation approaches into direct manipulation approaches, relational approaches, graph-transformation-based approaches, structure-driven approaches, and hybrid approaches [16].

The analysis of existing approaches revealed several limitations, such as the complexity in case of graph-transformation-based approach, and the limited applicability of structure driven approaches. However relational Approaches showed a high effectiveness in balancing between flexibility and declarative expression [16]. This category groups declarative approaches where the main concept is mathematical relations (like declarative approaches and mapping rules).

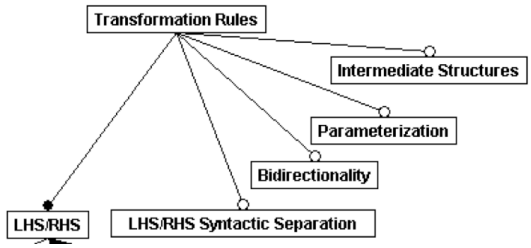


Figure 2 Features of transformation rules [16].

However, despite its advantages compared to other approaches, UI model-based approaches addressing adaptation suffers from complex transformation engine. This complexity

limits their extensibility and ability to incorporate increasingly acquired adaptation knowledge. Research in [4] highlights the traceability as a requirement for reuse and evolution. In order to overcome this shortcoming, we present a new relational transformation approach aimed at specifying transformation as a composition of clearly separated concerns

Adapt-first transformation approach addresses the requirement for reducing complexity by separating transformation concerns. In what follow, we present Adapt-first approach dealing with transformation complexity via separating adaptation and mapping concerns.

III. ADAPT-FIRST APPROACH

In this section, we present Adapt-First approach supporting transformations in model-based user interfaces development. The transformation is illustrated through different examples from USIXML project.

A. Overview

As we are using UsiXML models, it would be appropriate to give a slight background of UsiXML before we go further. UsiXML contains a set of models: Task model, Abstract User Interface (AUI), Concrete User Interface (CUI), Final User Interface (FUI), Resource Model, Context Model and others. These models are aimed to, when employed together, design an interactive UI that can be transformed into an executable final UI. More information can be found on the official site: www.usixml.org. UsiXML follows the CRF which represents a framework to define the design steps needed to describe a UI, including the features: Multi-level abstraction, Modality independence, among others.

The main principle in the adapt-first approach is to perform all the adaptations on the source model creating an intermediate model, and then transform the intermediate model to the destination model. The approach overview is depicted in figure 3.

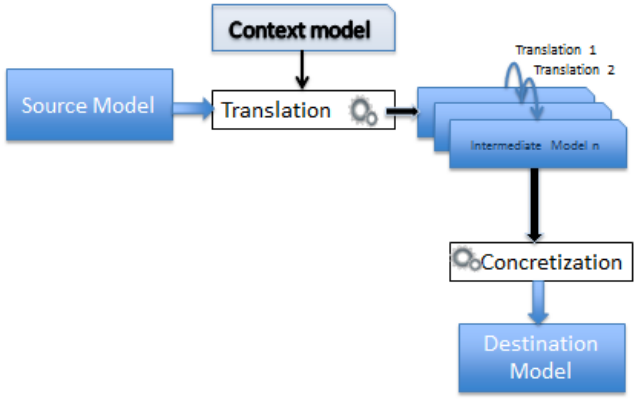


Figure 3. The Adapt-First approach

Adapt first capitalize on two main phases. the first represent an horizontal transformation (translation) this ensure the adaptation of the source model to the context of user the of this phase is an enriched model (intermediate) the second phase is a concretization from the adapted intermediate model to an adapted target model.

The translation might be cascaded, in that several translations can be run to reach the final intermediate model (the adapted model). This cascading can be fruitful to prioritize adaptation rules: what rule should be executed before others?

B. The method to employ adapt-first

In the following, we enlist the steps needed to apply the adapt-first approach in order to support contextual adaptation during transformation and UI generation:

1- Classify: Classify the adaptation rule at the appropriate/desired level of application: task, AUI, CUI. For instance, adaptation rules that split/merge widgets could be placed on the AUI level. Although splitting can be made also at the CUI level, the UI designer may prefer to apply it at the AUI level because it is the first place a logical grouping of elements is defined. Presentation rules that consider the color, text and other properties of widgets are placed at the CUI level. These attributes are defined at this level since they rely more on a concretization of the UI.

2- Analyze: Analyzing the adaptation rule: understand exactly what input is needed from the source and context model. In addition to understanding exactly the implications of this adaptation rule on the destination model: what elements are affected and what are the changes. For instance, take the case of the adaptation rule to the environment: change the widgets colors to meet light conditions (day and night). This rule is classified at the CUI level because it affects the colors properties. It needs as input CUI widgets and the context of use (day or night). It modifies the color property in the destination model (the FUI). Another example is the choice of interaction unit (the UI widget) based on the screen size. On small screens, a drop-down box might be favored over a list box to reserve the space. This adaptation is classified at the CUI level and affects the type of UI widget to choose.

3- Establish the intermediate meta-model:

   a. The intermediate model is specified within the same abstraction level of the source model. It consists on a contextualized version providing more information about the UI description. Additional attributes could be added to provide an enriched version.

   b. Abstract the implications of rule(s) in the intermediate model. The starting meta-model of the intermediate model can be the meta-model of the source model. For instance, to incorporate the adaptation rule on colors, the intermediate model can use the same meta-model for the CUI model (the source). Implications of this rule are reflected on the intermediate model by changing the color

attribute of the concerned widgets.

c. The intermediate meta-model might be modified to incorporate the adaptation rule. This is the case when the source model and the destination model don't use the same concepts, or when the destination model employs a wider point of view than the source model. For instance, adaptation to Arabic culture requires changing the reading direction of the screen to right-to-left [17]. CUI model doesn't support the direction attribute in UsiXML. On the other side, Java (as an example of the FUI) supports this attribute on UI widgets. This direction attribute can be added to the intermediate model (a change on the meta-model). The result of applying the language adaptation rule should affect this newly-added attribute in the intermediate model. We present another example later on the need to modify the intermediate meta-model.

d. The decision to modify the intermediate meta-model should be wisely taken considering simplifying the concretization transformation. The simplest form for concretization is to a mapping from the source to the destination. This means, modifications to the intermediate model should be meaningful to the destination model and clear enough to directly pass from the intermediate model to the destination.

4- Design the translation:

a. Cascade translations: Not all adaptation rules affect directly the destination model. For instance, users may perform the same task differently on a desktop or a mobile device. This type of adaptation employs an intermediate model that has the same task meta-model. Such rules should be applied at first. Adaptation rules that affect the destination model directly should be conducted later, resulting in another translation from the intermediate model to another intermediate model, like in figure below. Thus, Translation could be cascaded in order to prioritize adaptation rules.

b. If an adaptation rule depends on the result of another adaptation rule, cascading employed.

5- Design the concretization: the concretization should be straight forward to keep it simple. A transformation decisions should be taken based on: the element we are transforming, the parents of the entity in the source model, and the already-generated entities in the target model. This serves the purpose of extending the transformation engine easily when the input/intermediate model is enriched or modified.

C. *Illustrations*

We will present two examples to clarify the new approach.

First Example: Generating AUI from Task Model

Tran et al [18] explained an algorithm to transform task model to AUI based on weights given to tasks depending on their type and the weight given to the targeted platform (weight is calculated based on screen dimensions, processing power, memory, screen type…). Based on these weights, the algorithm chooses the appropriate grouping of tasks to be transformed into the same AUI container. We note the coupling between adaptation and the concretizations in the transformation process.

Let's apply the adapt-first method on the above case.

1- Classify: AUI containers should consider the targeted platform. The same task can generate different AUI containers according to the platform. we choose to apply this rule on the task model because it is the origin for AUI containers.

2- Analyze: the rule considers the task elements in the source model and group them according to the targeted platform. It affects the generation of AUI containers in the destination model.

3- Establish the intermediate model: the algorithm proposed by Tran determines the groups of tasks. we can then modify the intermediate meta-model (which is the task meta-model) to add the attribute "AUIContainer" on the task entity. This attribute will contain the result of the execution of Tran's algorithm. This property can be used easily in the concretization of the intermediate model.

4- Design the translation: for this rule, there is no need to cascade the translations. Anyway, tasks might be conducted by end-users differently on a mobile application or on a desktop. If we want to incorporate such rule in the adaptation, the task model should be adapted before generating the AUI. This can be achieved by cascading translations: from task model to an intermediate model (task model) and then to our proposed intermediate model in step 3.

5- Design the concretization: part of the algorithm of Tran proposed to generate AUIs from the task model using mapping rules. This part is simple enough to concretize the intermediate into the AUI model.

Figure 4 demonstrates how the transformation is simplified using the adapt-first approach. The figure shows how the adaptation is separated from the concretization. It also shows the modification to the task meta-model to add the attribute "AUIContainer" (values for this attribute are depicted by different colors). Concretization employs this attribute to generate AUI containers.

If we prefer to employ another algorithm to generate AUI containers from the task model, like the "Presentation Task Set" algorithm which is implemented in the TERESA tool [19]. We need to adapt the algorithm to reflect the results into the

newly-added attribute. The concretization part won't be affected.

The intermediate model provides also a way for incorporating customization and user-involvement in the adaptation. If the UI designer needs to manually tweak the adaptation, he can manually change the intermediate model. This can also be incorporated in a model-driven engineering approach by adding the customization as a rule. Customization rules could be incorporated at the ending stage of the translation step, or after completion of each translation step when cascading intermediate models.

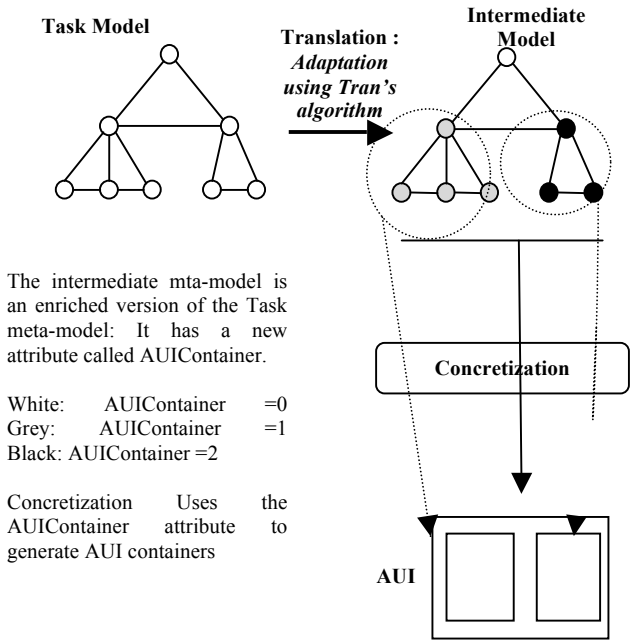


Figure 4. Generating AUI from Task using Tran's algorithm following the Adapt-First Approach .

Second Example: Adaptation to Culture

One of the important adaptations to culture is adapting the reading and writing direction to the user's language. Supporting reading and writing direction in UsiXML (the CUI model) was discussed in details in [17], where authors propose to add a new property to UIComponent element that holds the reading and writing direction of that UI component. Authors also explained the semantics regarding transformation support for this property. In brief: this property is inherited. The recursive algorithm to calculate the value is: (1) if a UIComponent has an explicit value, use it, (2) if not, use value of the parent. (3) If the root parent doesn't have a value, use the default value imposed by the culture.

Let's apply the adapt-first method on this adaptation to culture example:

1- Classify: According to CRF, AUI is platform-independent, modality-independent. The adaptation rule is dedicated to Graphical GUI, thus it is modality-dependent and affects the CUI model.

2- Analyze: the rule considers CUI elements in the source model and modify their reading and writing direction. The implications affect the reading and writing direction attribute (the "direction") on each graphical component.

3- Establish the intermediate model: the intermediate meta-model is the same as the CUI meta-model. There is no need for modifications.

4- Design the translation: for this rule, there is no need to cascade the translations.

5- Design the concretization: map a CUI element type with the appropriate value for the directions with the corresponding FUI element. Depending on the FUI model and its support for different reading and writing directions, the mapping can be configured to either generate a different FUI component or change an attribute in the component.

IV. IMPLEMENTATION

The implementation we provide here is a proof of concept of the approach. We demonstrate how to apply the approach on the CUI model. A case study on "car rental" UI is demonstrated. The case study shows how to adapt the UI to culture and adaptation to screen size following the adapt-first approach.

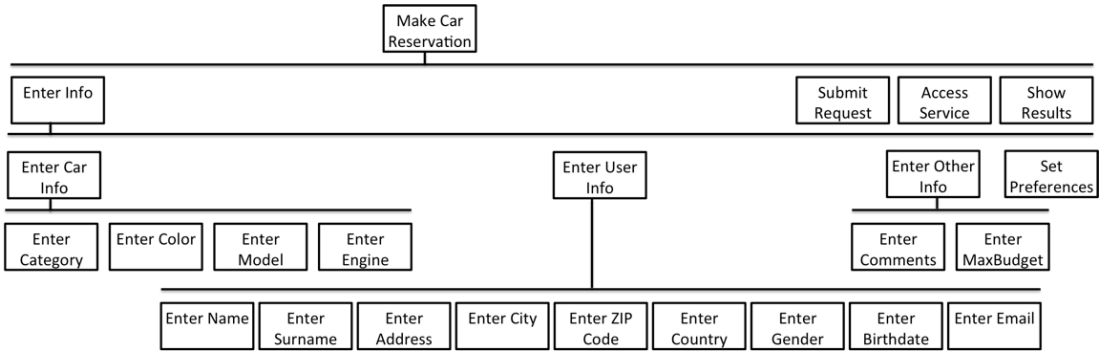
Firstly, we describe the case study, the architecture and then explain how adaptation is implemented.

A. Overview

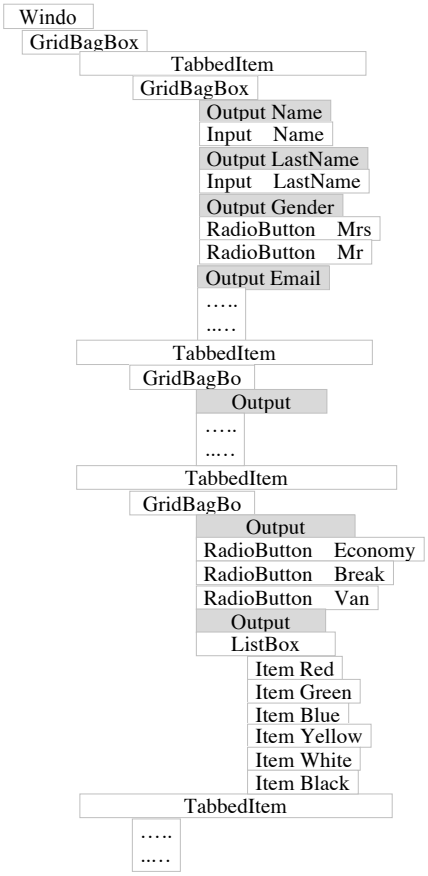
Car Rental is a UI for a client willing to rent a car. The UI allows the user to determine his options. The specification of the case study is:

"The car rental case study consists in a scenario in which the interactive system supports users in the task of renting a car. In this sense, various context information can be used to adapt application aspects, and to properly display the list of cars to rent, enabling users to make choices and to accomplish the main task."

Figure.5 illustrates the hierarchical task analysis (HTA) for the car rental case study. Basically, users must provide information about the car (i.e. category, color, model, and engine), then their own information (i.e. name, surname, address, city, ZIP code, country, gender, birthday and email), and finally other information (i.e. commentaries, and maximum budget). Once the preferences are set and the request is submitted, users access the service and the results. To conclude the rental, users select a car, and define the period of interest. This task model is merely illustrative, and serves as a basis for the implementations, mainly because some of the context-aware adaptations envisaged affect the tasks' sequence, so specializations of this model are expected. The figure illustrates a simplified representation of the CUI focusing on key elements for this case study. The CUI is described using an XML file. The figure also shows an excerpt of this xml for illustration purposes. The FUI uses Java Swing to render the CUI model.



The case study task model



A simplified Illustration of CUI model

Figure 5. Car Rental: The hierarchical task analysis

This case study handle diverse adaptation scenarios, we present in this section three adaptation rules:

1- Adaptation to reading and writing direction of the user's language.

2- Translation of text and resources according to the user's language.

3- Adaptation to large and small screen. On small screens, replace the list box (depicted in figure 4) by a combo box to reserve space.

The transformation tool from CUI to FUI is developed following the adapt-first approach. It recognizes the separation between the translation and the concretization.

B. Implementing the translation:

Translation is concerned with adaptation to context of use and implementing adaptation rules. For this purpose, the rule-engine tool: the Java Expert System Shell (Jess) [20]. Jess leads the adaptation based on Event-Action rules. Actions manipulate the intermediate model for adaptation.

Jess engine was integrated in the translation as shown in Figure 6. The UsiXML file is parsed and loaded in the memory, then the CUI model is parsed and each element is injected in Jess rule engine as a fact. The other supporting models: resource, context are injected as facts equally. The adaptation starts explicitly by the "adapt" event where jess rules would be fired according to elicited context of use facts. The resulting model is the intermediate model, which represents the adapted CUI model.

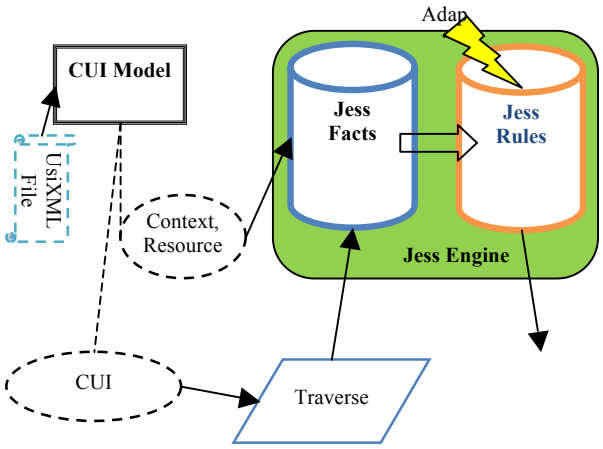


Figure 6. The translation Design

C. Implementing the concretization

The concretization engine follows a simple architecture, where it has as input the intermediate model and the mapping schema that is used to resolve the generation of each element in the input model to the corresponding element in the destination model. Figure 6 shows the architecture of the concretization part. It has two main components: the Cui Model Traversal, which simply traverses the adapted Cui model elements (the intermediate model), and the widget selector which generates the corresponding element in the target model according to the mapping schema stored in the file (map.xml). The mapping file is an xml file that relates an element from the Cui model with a snippet of code that will be executed under a specific condition. Figure 7 shows an example of generating Swing java code from the "Window" Cui Element.

D. Implementing adaptation rules

To adapt to the reading and writing direction, a rule was implemented in Jess to set the "direction" attribute on every CUI component to "Right To Left" if the language is Arabic and to "Left To Right" if the language is English. The intermediate model is guaranteed to have a value for this attribute on every CUI component, which facilitates the work of the concretization later by avoiding the recursive calculation for this property (explained in the previous section).

To translate the text, another rule was added into Jess. This rule modifies the display text of each CUI component by the appropriate translation per to the user's language.

Adaptation to screen size requires changing the list box in the "preferences" tab into a combo box to preserve space. This is achieved by adding another rule to Jess.

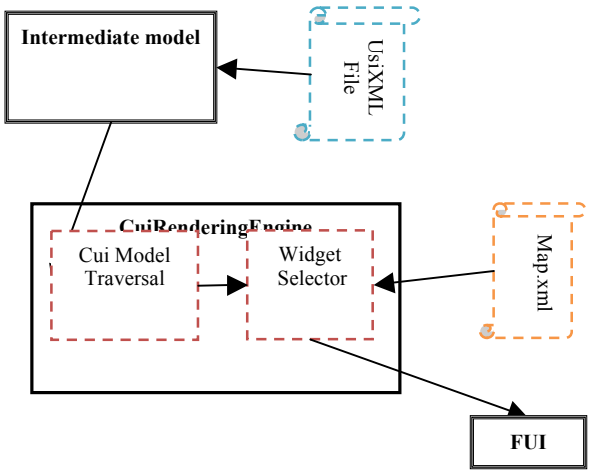


Figure 7. The concretization design.

Figure 8 depicts part of the transformation from CUI to FUI using the approach and the tools. Due to space limits, we introduce only part of the case study: the preferences tab of the main screen. The figure shows the CUI model in the initial state (as designed). Adaptation is performed cascaded into two translations: the first adapts to the screen size and the second adapts to culture. This cascading is to prioritize the three adaptation rules employed in the case study. This cascading helps to ensure that the replaced combo-box is well-adapted to language by executing the adaptation to language rules after the replacement. Changes imposed by adaptation rules are depicted in the figure in red color. We also demonstrate the FUI and changes after each adaptation.

V. CONCLUSIONS AND FUTURE WORKS

We presented in this paper the adapt-first approach. It consists of a set of concepts (model translation transformation, intermediate model, model concretization transformation) a method (adapt then transform) and tools to support this approach. The purpose of this approach is to simplify the transformation engine which enhances ability to incorporate further adaptation rules. This is achieved by separating the translation (the adaptation to the context of use) from the concretization: transforming the model to a more concrete

model (like from CUI to FUI).

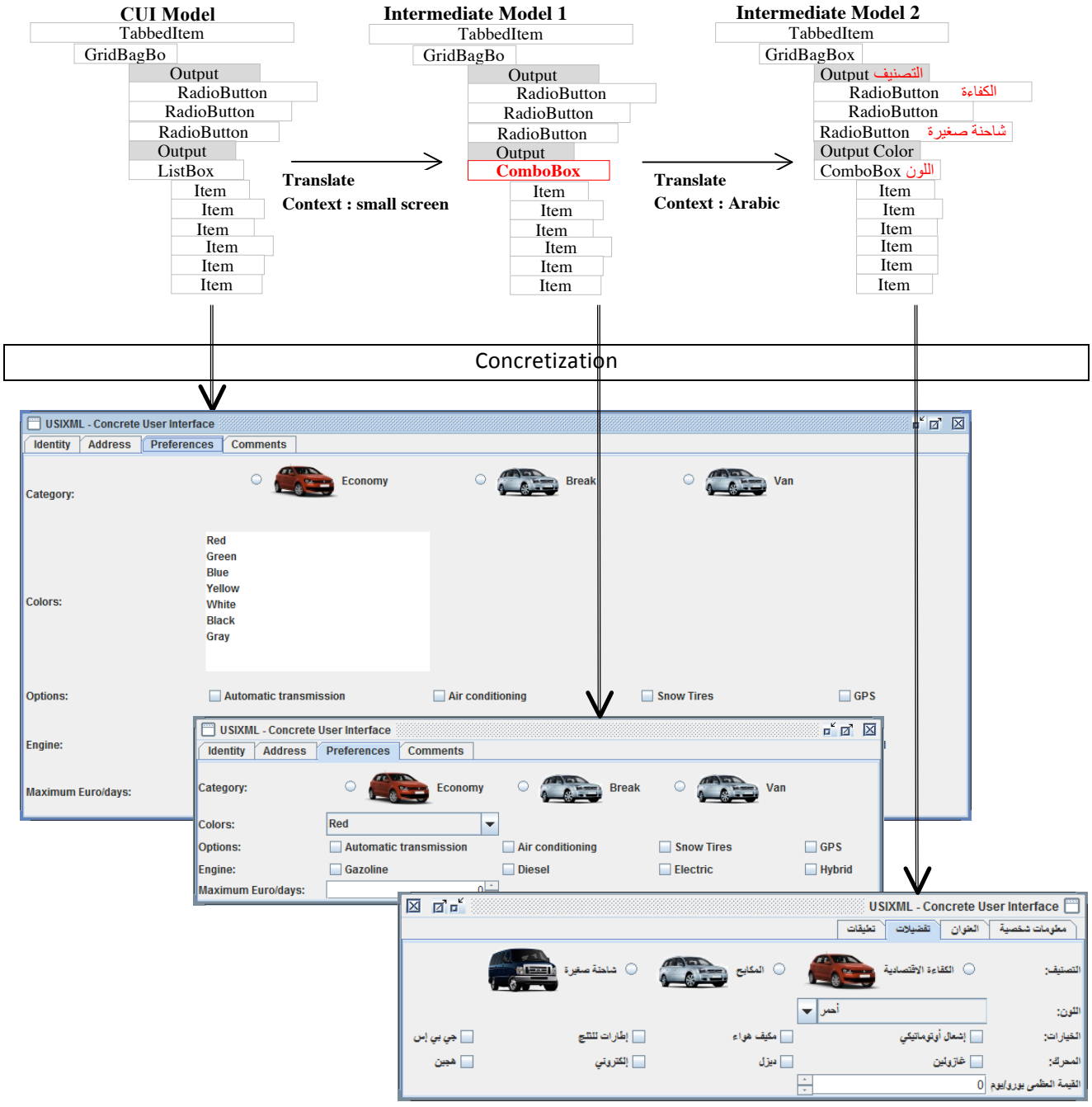


Figure 8. Adapt-first approach applied on the “Preferences” tab of the case study

The case study shows that a simple mapping concretization can be created that is independent of the adaptation. The intermediate meta-model might be modified to support adaptation rules. A wise design of these modifications while and considering the implication on the concretization can lead to manageable changes on the concretization part.

The complexity of the context of use and implication on the UI model are eased by employing the intermediate model, and scaled/prioritized by employing cascading intermediate models.

More work on the tools is needed to enhance customize of adaptation and user feedback. One possible direction is to create a “beautification tool”. The beautification tool would allow the UI designer (and maybe the end-user) to override the

result of an adaptation rule by explicitly/manually tweaking the concerned elements in intermediate model. It should enable also enabling/disabling rules. On another side, the beautification tool would allow managing priorities of rules, thus controlling the cascading translation.

REFERENCES

[1] Hutchinson, J., Whittle, J., Rouncefield, M., and Kristoffersen, S. (2011) Empirical assessment of MDE in industry, Proceedings of the 33rd International Conference on Software Engineering ICSE'2011, ACM Press, New York, pp. 471-480.

[2] http://www.w3.org/TR/mbui-intro/

[3] Pilemalm, S., Hallberg, N., Sparf, M., and Niclason, T. (Dec. 2012) Practical experiences of model-based development: Case studies from the Swedish Armed Forces, Systems Engineering , Volume 15 Issue 4, pp. 407-421.

[4] Mezhoudi N, PerezMedina JL, and Vanderdonckt Jean. Towards a Conceptual Model for UIs Context-Aware Adaptation. In proc. the 2nd World Congress on Multimedia and Computer Science, January 2015, International Conference on Computer Information Systems. ISBN: 978-9938-9563-2-0.

[5] Motti, V. G., & Vanderdonckt, J. (2013, May). A computational framework for context-aware adaptation of user interfaces. In Research Challenges in Information Science (RCIS), 2013 IEEE Seventh International Conference on (pp. 1-12). IEEE.

[6] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers , 15(3): 289–308.

[7] Bouillon, L., Vanderdonckt, J., Retargeting Web Pages to other Computing Platforms with Vaquita, Proc. of IEEE Working Conf. on Reverse Engineering WCRE'2002 (Richmond, 28 October-1 November 2002), A. van Deursen, E. Burd (eds.), IEEE Computer Society Press, Los Alamitos, 2002, pp. 339-348.

[8] Akiki, Pierre A.; Bandara, Arosha K. and Yu, Yijun (2015). Adaptive model-driven user interface de- velopment systems. ACM Computing Surveys, 47(1) (In press).

[9] Criado, J., Iribarne, L., Padilla, N., Troya, J., and Valle- cillo, A. An mde approach for runtime monitoring and adapting component-based systems: Ap- plication to wimp user interface architectures. In Euromicro Conference on Software Engineering and Advanced Applications. 2012.

[10] Paterno,F. Santoro,C. Mantyjarvi,J. Giulio Mori, Sansone,D. Authoring pervasive multimodal user interfaces, Int. J. Web Eng. Technol.2008, 235–261.

[11] Gajos, K. Z., Czerwinski, M., Tan, D. S., and Weld, D. S. 2006. Exploring the design space for adaptive graphical user interfaces. In Proc. of the Conf. on Adv. Visual Int.AVI '06.ACM, NY,USA,201-208

[12] Blumendorf, M., Lehmann, G. and Albayrak, S. 2010. Bridging Models and Systems at Runtime to Build Adaptive User Interfaces. Proceedings of the EICS '10

[13] Bodart, F. Hennebert, A.M. Leheureux, J.M. Provot, I. B. Sacre, J. Vanderdonckt, Towards a systematic building of software architectures: The TRIDENT methodological guide, in: Design, Specification and Verification of Interactive Systems, Springer, Wien, 1995, pp. 262–278.

[14] Mitrovic , N. Royo, J.A. and. Mena, E. Performance Analysis of an Adaptive User Interface System Based on Mobile Agents.In Handbook of Research on User Interface Design and Evaluation for Mobile Technology, 2007.

[15] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., & López-Jaquero, V. (2005). USIXML: a language supporting multi-path development of user interfaces. In Engineering human computer interaction and interactive systems (pp. 200-220). Springer Berlin Heidelberg.

[16] Czarnecki, K. Helsen, S. Classification of Model Transformation Approaches. Workshop on Generative Techniques in the Context of Model-Driven Architecture, OOPSLA'03. 2003.

[17] Khaddam, I. and Vanderdonckt, J. Adapting UsiXML User Interfaces to Cultural Background. In Proc. of 1st Int. Workshop on User Interface eXtensible Markup Language UsiXML'2010 (Berlin, 20 June 2010). Thales Research and Technology France, Paris (2010), pp. 163–170.

[18] Tran, V., Vanderdonckt; J., Tesoriero R., Beuvens, F. (2012). An Analytical Benchmarking of Algorithms for Generating Abstract User Interfaces, EICS'12. Copenhagen, Denmark, June 25–28, 2012. pp. 101-110.

[19] Berti, S., Correani, F., Paterno, F., Santoro, C., The TERESA XML language for the Description of Interactive Systems at Multiple Levels. In: Proc. Of the Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages, pp. 103-110 (2004).

[20] Hill, Ernest Friedman. Jess in Action: Java Rule-Based Systems. Manning Publications Co., Greenwich, CT, USA, 2003.