Transformations Between Specifications of Requirements and User Interfaces

Sevan Kavaldjian, Hermann Kaindl
Institute of Computer Technology
Vienna University of Technology, Austria
{kavaldjian, kaindl}@ ict.tuwien.ac.at

Kizito Ssamula Mukasa
Fraunhofer Institute for Experimental Software Engineering (IESE)
Germany
kizito.mukasa@ iese.fraunhofer.de

Jürgen Falb
Institute of Computer Technology
Vienna University of Technology, Austria
falb@ict.tuwien.ac.at

ABSTRACT
Separating requirements from user interface specifications often leads to unusable systems or to systems that do not support the users' needs. We address this issue by introducing explicit transformations between models of these different "worlds". In fact, we show how to transform artifacts in a model specifying requirements to artifacts in a model specifying a user interface, and vice versa (inverse transformations). We also transform from a more abstract to a more concrete model of a user interface, and vice versa. In effect, this allows starting either from requirements and getting support for transforming to a user interface, or from a user interface prototype and getting support for transforming to requirements.

INTRODUCTION
Usually, model transformations lead from higher-level to lower-level models and program code in software development, e.g., from architectural to design models. Model-driven generation of user interfaces (UIs) typically leads from higher-level task models to UI models (at certain levels of abstraction) and a final UI. Unlike other approaches, we also transform between models on the *same* abstraction level. So, in addition to operationalizing as usual, we make use of artifacts in one "world" for creating related ones in another. Additionally, we allow transformations back and forth between the same pair of models.

We present this approach of bidirectional transformations between models of the same level of abstraction in the context of our previously defined Requirements Specification Language (RSL) [3]. In contrast to most other languages for requirements specification, RSL is a language that integrates requirements with UI specifications [5]. This integration along the representation dimension is supposed to facilitate combined work on requirements and user interfaces along the process dimension as well. In particular, we developed transformations between artifacts of requirements and UI specifications within RSL.

The remainder of this paper is organized in the following manner. First we sketch those parts of the RSL Metamodel needed within the scope of this paper. Then we present our overall approach of transforming in this context. For elaborating on this overall approach, we specify (MOLA[1]) transformation rules from the requirements specification to an abstract UI specification, and vice versa. In addition, we specify such rules for transforming from an abstract UI specification to a more concrete UI prototype. Finally, we compare our approach with related work.

BACKGROUND
In order to make this paper self-contained, we need to sketch some background material about RSL. In particular, we explain those parts of its metamodel for specifying requirements and UI that we build our transformations upon.

Scenarios and use cases are popular in requirements engineering these days. Therefore, RSL includes ConstrainedLanguageScenarios contained in Use Cases. These provide a textual representation and consist of a sequence of SVO sentences describing the flow of interaction between the user and the system to be developed. An SVO sentence has a Subject and a Predicate, which in turn has one Verb, and an Object. Each word used in a sentence can be mapped to its specific meaning in the domain vocabulary. It is possible to define the meaning of each word in this context, also by reusing terminology from WordNet [7].

For specifying a UI, RSL provides generic elements that make it possible to define a UI independently of modality and toolkit. The static aspects of the UI, i.e., its structure and layout, can be described by using elements like InputUIElement for data input; TriggerUIElement for triggering actions; SelectionUIElement for exclusive or non-exclusive selection from more than two options; and UIContainer or UIPresentationUnit, which are used as containers of other UI Elements.

[1] `http://mola.mii.lu.lv/`

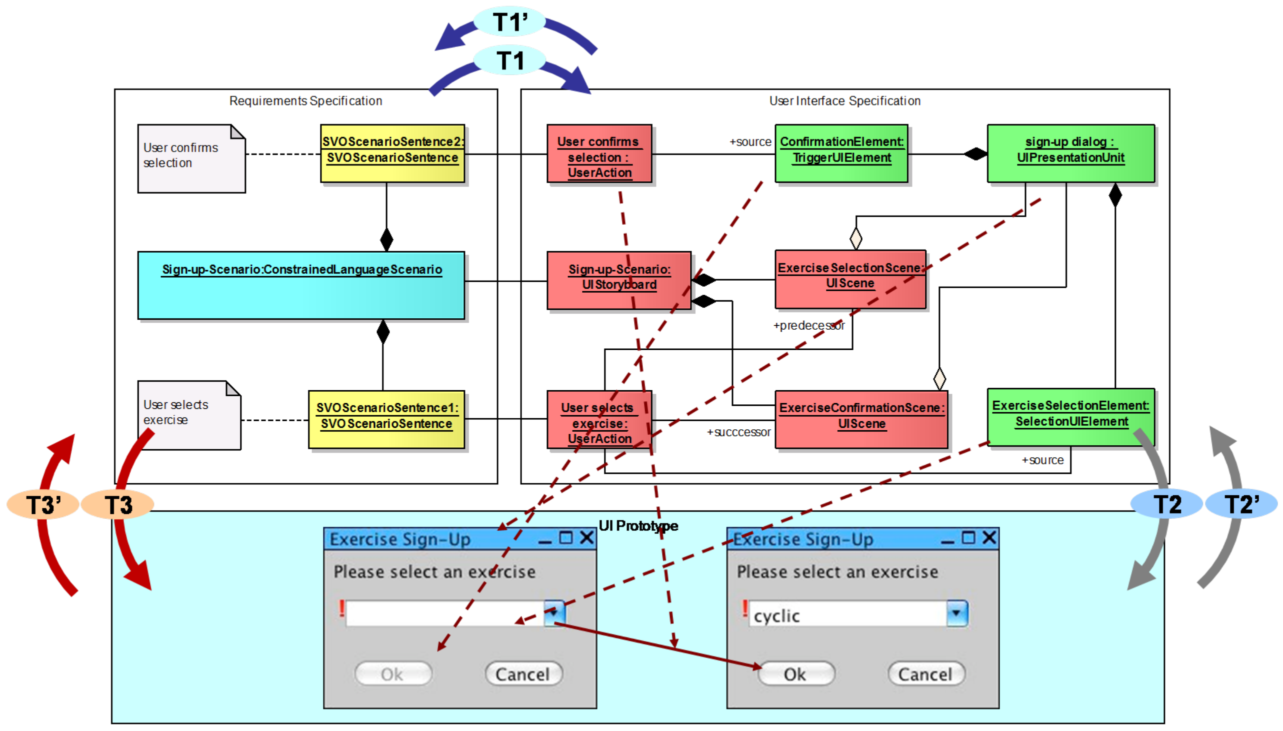
T1'
T1
Requirements Specification
User confirms selection
SVOScenarioSentence2: SVOScenarioSentence
Sign-up-Scenario:ConstrainedLanguageScenario
User selects exercise
SVOScenarioSentence1: SVOScenarioSentence
User Interface Specification
User confirms selection : UserAction
+source
ConfirmationElement: TriggerUIElement
sign-up dialog : UIPresentationUnit
Sign-up-Scenario: UIStoryboard
ExerciseSelectionScene: UIScene
+predecessor
User selects exercise: UserAction
+succcessor
ExerciseConfirmationScene: UIScene
ExerciseSelectionElement: SelectionUIElement
+source
T3'
T3
T2
T2'
UI Prototype
Exercise Sign-Up
Please select an exercise
Ok
Cancel
cyclic

Figure 1. Overview of specifications and their transformations.

The dynamics of the UI, i.e., the behavior related to user interaction, can be described by using RSL elements like UIStoryboard, UIScene, and UserAction. A UIStoryboard is a series of scenes displayed in a sequence. The "presentations" of the individual scenes are defined in UIScenes. UIScenes can be connected by UserActions indicating the triggering action of the user. A UserAction is performed on one source UIElement in a predecessor UIScene and can result in a transition to a successor UIScene as well as influence some of its UIElements.

Since these UI elements are modality-independent, they specify an *abstract* UI (according to [1]), that can be used for the different modalities found in advanced user interfaces. In order to make a UI specification more understandable for a user, a *concrete* UI [1], which defines the modality, is better suited. Therefore, RSL also includes related elements, whose concrete syntax can even serve for specifying a UI Prototype, see the bottom of Figure 1.

OUR OVERALL APPROACH OF TRANSFORMING

Based on the RSL language and its metamodel, let us illustrate our overall approach of transforming between requirements and UIs, more precisely their specifications in the form of models.

Assume that a requirements engineer has specified a scenario using SVOScenarioSentences [3]. Instead of having a UI designer manually create a related UI specification, transformation rules according to T1 in Figure 1 may be applied. They would lead to a partial model of an abstract UI. Applying transformation rules according to T2 would then lead to part of a more concrete UI, i.e., a UI Prototype. Of course, this prototype is most likely not the user interface of the real application yet, but it may well serve the purpose of illustrating the textual scenario in the requirements specification to users. When concrete scenarios are directly used for UI design, they may well lead to bad UIs, since they may induce an interaction approach defined ad hoc in a different context. So, such scenarios should be made more abstract first to capture the essence [2].

We may also assume that it goes the other way round. Users themselves or together with a UI designer have developed a prototypical storyboard, already using the GUI modality in its concrete syntax. This approach facilitates their intuition of how a system to be built may be used. For developing such a piece of software, however, a requirements specification may still be needed or simply useful for the developers. Instead of having a requirements engineer manually write such a specification, transformation rules according to T2' in Figure 1 may be applied. They would lead to a partial model of an abstract UI, this time based on the UI Prototype. Applying transformation rules according to T1' would then lead to a part of the requirements specification, a scenario. Again, the generated scenario might not be in the final version yet, but it is consistent with the UI Prototype.

TRANSFORMATIONS FROM REQUIREMENTS SPECIFICATIONS VIA UI SPECIFICATIONS TO UI PROTOTYPES

Let us illustrate now what transformation rules are needed to transform our concrete example in Figure 1. It consists of a ConstrainedLanguageScenario composed of the following two

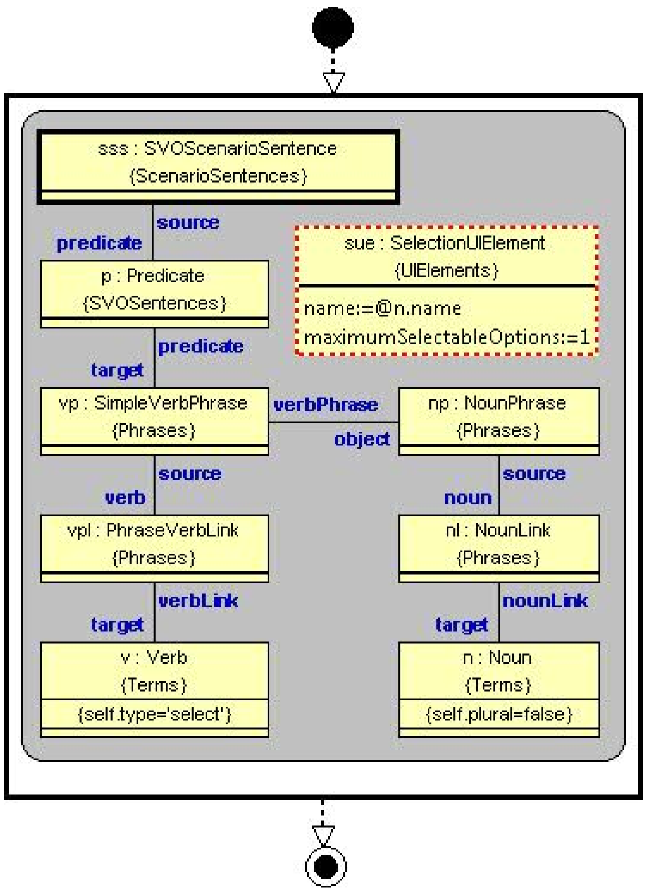

Figure 2. MOLA Rule R1 for T1.

SVOScenarioSentences: 1. "User confirms selection." and 2. "User selects exercise."

The following rules implement the transformation T1:

Rule 1: Each SVOScenarioSentence whose verb implies "selection" is transformed into a SelectionUIElement. Its name is the Noun in the VerbPhrase (see [3] for the definition of VerbPhrase).

Rule 2: Each SVOScenarioSentence whose verb implies "triggering an action/event" is transformed into a TriggerUIElement. Its name is the Noun in the VerbPhrase.

Rule 3: Each SVOScenarioSentence whose verb implies "showing information" is transformed into a UIPresentationUnit. The name of this UIPresentationUnit is the Noun of the Object in the VerbPhrase.

Note, that these rules both require and utilize a mapping between synonyms. We reuse this mapping from WordNet.

Rule 4: Each ConstrainedLanguageScenario is transformed into a UIStoryboard of a similar name.

Rule 5: Each SVOScenarioSentence in a ConstrainedLanguageScenario is transformed into a UIScene. Its sceneDescription is the text of the Sentence and its sceneNumber equals the seqNumber of that sentence. The UIScene is linked to the UIPresentationUnit corresponding to this sentence.

Rule 6: Each SVOScenarioSentence whose subject is not "System" is transformed into a UserAction. Its source is the UIElement (other than the UIPresentationUnit) associated with the sentence. The predecessor of this UserAction is the UIScene of the previous sentence and its successor is the UIScene of the current sentence.

Applying these rules to our example results in the UI Specification Model in the right box of Figure 1. The rule execution order has no impact on the generated model.

Figure 2 illustrates the formalized Rule 1 for T1 in MOLA, composed of the following elements. The outer bold rectangle symbolizes a *for-each* loop. The rounded rectangle inside represents the actual rule, that will be repeated for each matched element. The small boxes inside the rule represent different kinds of classes, depending on their borderline style. When the thickness of the border line is regular, they represent a "normal" class. A bold border lined box represents a loop variable. A dashed lined box border represents a class that will be created by the transformation rule. The small black circle represents the starting point of the rule. The double rounded circle represents the end point of the rule. In particular, Rule 1 iterates over all SVOScenarioSentences. Whenever the type of the verb is "select" and the noun is not plural, the rule matches. In this case, a SelectionUIElement of the UI Specification Model is generated. The name attribute is set to the name of the noun and the maximumSelectableOptions attribute is set to 1.

The following rule is one example of transformation T2:

Rule 7: Each SelectionUIElement that only allows the selection of one Option is transformed into a Class associated with the Combobox Stereotype. The name of the class is composed of the name of the SelectionUIElement and "ComboBox".

Figure 3 illustrates the formalized Rule 7 for T2 in MOLA. It iterates over all SelectionUIElements where the attribute maximumSelectableOptions is set to 1 in the UI Specification and creates one class each. The name attribute of the class is set to the SelectionUIElement name to which the suffix "ComboBox" is added. It also creates an association with the corresponding stereotype.

TRANSFORMATIONS FROM UI PROTOTYPES VIA UI SPECIFICATIONS TO REQUIREMENTS SPECIFICATIONS

This section introduces transformations for T1' and T2', from UI Prototypes via UI Specifications to Requirements Specifications.

The following rule for T1' represents the exact opposite to Rule 1 in T1:

Rule 1': Every SelectionUIElement with the attribute maximumSelectableOptions set to 1 is transformed into an SVOScenar-

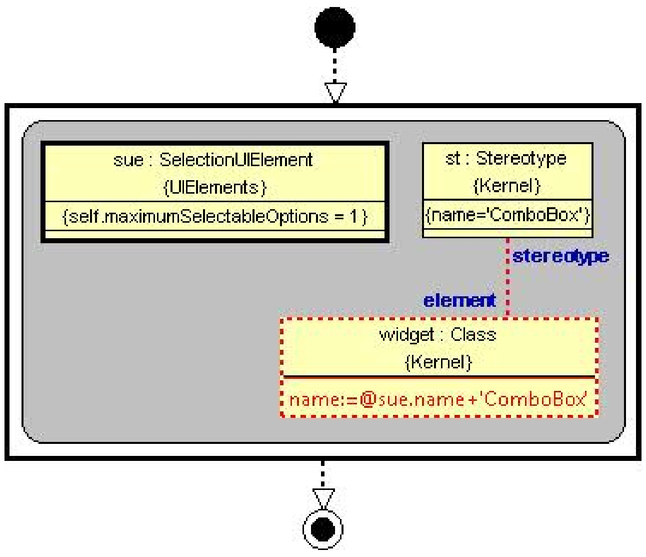

Figure 3. MOLA Rule R7 for T2.

ioSentence. The Verb "select" is the representative of the equivalence class for selection. The noun is the value of the name attribute of the SelectionUIElement.

This rule looks quite similar to Rule 1 in T1, but there is a subtle difference with the inverse transformation, since Rule 1' always transforms to the verb "select". If another verb of the same type is used in the source model of T1, e.g., "chose", then applying T1 and subsequently T1' will not result in the identical model. This is more of theoretical than of practical interest, since transforming back and forth identically is not needed for applying this approach.

The following rule for T2' represents the exact inverse to Rule 7 in T2:

Rule 7': Every class with the suffix "ComboBox" in the name attribute is transformed into a SelectionUIElement with the maximumSelectableOptions attribute set to 1.

The transformation T3 can be seen as a composition of transformation rules for T1 and T2. The transformation T3' can be seen as a combination of the rules for T1' and T2'.

RELATED WORK

Mori et al. [4] introduced ConcurTaskTrees (CTT) as task models, from which they can derive and semi-automatically generate UIs. Their approach leads from higher-level to lower-level models (like our T2 transformation). Our approach as presented in this paper does not start from high-level task models, but from detailed requirements specifications. Alternatively, our approach can start from concrete UIs and lead automatically to artifacts in a requirements specification.

Panach et al. [6] propose a method to bring Software Engineering (SE) and Human-Computer Interaction (HCI) closer together, much as we do with our transformation approach. They capture interactions with sketches and transform them into structural patterns of CTT. Similarly to our approach, they transform up and downwards in the reference framework [1]. In addition, we transform horizontally (on the same level of abstraction) and between different "worlds".

CONCLUSION

This paper presents a model-driven approach to transformations from requirements to UI specifications and vice versa. We are not aware of any transformation approach between these different "worlds", which are partly even on the same level of abstraction. And it is new to have transformations back and forth between the same pair of models.

This approach of explicit transformations may help to bridge the usual gap between separated requirements and UI specifications. It may also make the overall development more efficient, since it makes explicit use of artifacts from any one "world" to create artifacts in the other one.

ACKNOWLEDGMENT

This research has been carried out in the ReDSeeDS project and is partially funded by the EU (contract number IST-33596 under the 6th framework programme), see `http://www.redseeds.eu`.

REFERENCES

1. G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting With Computers Vol. 15/3*, pages 289–308, 2003.
2. L. Constantine and L. A. D. Lockwood. *Software for Use*. ACM Press, New York, NY, 1999.
3. H. Kaindl, M. Śmiałek, D. Svetinovic, A. Ambroziewicz, J. Bojarski, W. Nowakowski, T. Straszak, H. Schwarz, D. Bildhauer, J. P. Brogan, K. S. Mukasa, K. Wolter, and T. Krebs. Requirements specification language definition. Project Deliverable D2.4.1, ReDSeeDS Project, 2007. www.redseeds.eu.
4. G. Mori, F. Paterno, and C. Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Transactions on Software Engineering*, 30(8):507–520, 8 2004.
5. K. Mukasa and H. Kaindl. An integration of requirements and user interface specifications. In *Proceedings of the Sixteenth IEEE International Requirements Engineering Conference (RE'08)*, September 2008.
6. J. I. Panach, S. Espana, I. Pederiva, and O. Pastor. Capturing interaction requirements in a model transformation technology based on MDA. *Journal of Universal Computer Science*, 14(9):1480–1495, 2008.
7. K. Wolter, M. Smialek, D. Bildhauer, and H. Kaindl. Reusing terminology for requirements specifications from WordNet. In *Proceedings of the Sixteenth IEEE International Requirements Engineering Conference (RE'08)*, September 2008.