Towards a Uniform Model Transformation Process for Abstract User Interfaces Generation

Lassaad Ben Ammar[a]
Prince Sattam Bin Abdul-Aziz University, Kharj, Riyadh, Saudi Arabia
University of Sfax, Sfax, Tunisia

Keywords: Unified Model Transformation, Model-based User Interface Development, Software Engineering.

Abstract: The Model-Driven Development (MDD) paradigm is currently being taken interest in the field of Software Engineering (SE). It allows simplification and automation of the development process by defining models and transformations of those models. Model-Based User Interface Development (MBUID) is a variant of the MDD paradigm in the domain of UI development. It aims to reduce the efforts needed to develop UIs. It has had a significant research by the SE community leading to the definition of the Cameleon Reference Framework (CRF) as a unifying framework which structures the UI development process. In the last few years, several research works have been conducted with the aim of proposing a Cameleon-compliant UI development process. This situation leads to a series of important shortcomings; among them we quote in particular the lack of consensus (standard) about the information the different models have to contain and how to transform these models. With the aim of solving this issue of giving a uniformed UI development process, this paper presents an initiative towards uniformed task model and its transformation to an abstract user interface.

1 INTRODUCTION

The development of user interfaces (UIs) has become one of the most important elements to consider when developing software applications (Molina et al., 2012). Model-based User Interface Development (MBUID) is widely recognized as a promising approach which permits the simplification and automation of the UI development process while reducing the developer's needed efforts (Meixner et al., 2011). It aims to identify high-level models that will undergo a series of transformations in order to (semi-) automatically generate the final UI.

In the last few years, the increase adoption of MBUID has led to an ever-increasing number of user interfaces development methods and techniques. A renowned work in this context is the Cameleon Reference Framework (CRF) (Calvary et al., 2003). It structures the UI development process into four levels of abstraction:

- Task & Concepts (T&C): describe the various user tasks to be carried out and the domain-oriented concepts as they are required by these tasks to be performed.
- Abstract UI (AUI): expresses the UI in terms of Abstract Interaction Units (AIU) or Abstract Interaction Objects (AIOs) (Vanderdonckt and Bodart, 1993) as well as the relationships among them.
- Concrete UI (CUI): concretizes an abstract UI for a given context of use into Concrete Interaction Objects (CIOs) (Vanderdonckt and Bodart, 1993) so as to define widgets layout and interface navigation. These CIUs are modality-dependent, but implementation technology independent.
- Final UI (FUI): expresses the UI in terms of implementation technology dependent source code. A FUI can be represented in any UI programming language (e.g., Java UI toolkit) or mark-up language (e.g., HTML).

Several different methods and models for UI development are proposed within the Cameleon framework (e.g. (Akiki et al., 2016) and (Molina et al., 2014)). This variety raises a particular problem for developing user interfaces via model-based approaches as different models and transformation

[a] https://orcid.org/0000-0002-4698-3693

Ben Ammar, L.
Towards a Uniform Model Transformation Process for Abstract User Interfaces Generation.
DOI: 10.5220/0007763905330538
In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*, pages 533-538
ISBN: 978-989-758-375-9

processes are exploited: they are not well integrated into a single consolidated conceptual framework that facilitates their usage by software developers. This paper attempts to solve this problem by introducing uniform process for generating abstract user interfaces. Note that the ultimate goal of the research project to which this paper belongs is to introduce a unified Cameleon-Compliant process for user interface generation. In this paper, the interest is focused on the first part of the development process aiming to generate an abstract user interface from a Task&Concept model.

In the remainder of the paper, we present an overview of the most significant (referenced) model-based approaches serving as a basis to provide our proposal for unifying the abstract user interface development process. Following this, a case study and a tool supporting the meta-model are presented. The paper is wrapped up by summarizing our work, deriving conclusions and addressing future work and challenges.

2 RELATED WORK

The aim of this section is to summarize the state of the art and the effort made in the field of user interface generation via model based approach. The focus is placed on some proposals that are considered relevant due to their wide citation in related works.

In (Tran et al., 2012), an algorithm is presented to generate systematically all potential abstract user interfaces from task and domain models. The engineering process entails 9 steps using different resources (models and documents) which are defined within the UsiXML framework. This makes the proposed approach closely related to that framework and prohibits its adoption by other researchers. Furthermore, the cost and performance of such an approach is the main weakness since the analyst must specify the transformation rules for all potential abstract user interfaces.

(Molina et al., 2012) proposed an interesting tool namely CIAT-GUI that allows to (semi-) automatically obtain the final graphical user interface of an information system from declarative models (a task model in Concur Task Trees CTT[2] notation and domain model in UML notation). This proposed work offers visual-design tools for the various levels of abstraction. Indeed, it presents a very interesting basic idea about the automatic process for user interface generation. However, several gaps and limitations still need to be addressed in this proposition. For example, the task model is analyzed several times in the development process even during the generation of the concrete user interface. This contradicts the principles of an MDE[3] development process which only consider the task model at the beginning of the development process (for generating the abstract user interface model). There are also gaps and limitations that pertain to the implementation details. This includes the analysis of the task tree in a bottom-up process starting from the leaf to the root instead of the reverse process (top-down). This can raise several questions about the cost/effectiveness of the implementation.

In (Limbourg et al., 2001), a series of representative task models are analysed and their meta-model are merged in a unified task meta-model. Several semantic mapping rules between individual task meta-models and the uniformed task meta-model are established in order to read and understand any potential task model towards its exploitation in a model-based approach. Gaps and limitations of this proposal are closely related to two main issues. The first one concerns the effort needed to consolidate a new meta-model by modelling their characteristics which are not presented in the unified meta-model. The second one pertains to the expressiveness of the unified meta-model since it considers only task models leaving aside relevant concepts from other models (e.g. domain model).

The MBUI incubator group of the W3C (MBUI, 2014) published two initiatives to uniform task model and abstract user interface model. These initiatives are interesting as for the concepts to be considered in each model. However, they are proposed with theoretical troubles disregarding the development perspectives, which may cause overhead to the application developers while implementing the transformation process.

Based on the aforementioned proposals, it has become clear that although there were multiple attempts to generate user interface within a model-based approach, several shortcomings still persist. Among them we mention:

[2] CTT: supports a hierarchical description of task models with the possibility of specifying a number of temporal relations among them

[3] MDE: Model-Driven Engineering is a recent software engineering approach aiming at the development of software system by considering model as primary artifact and their transformation from the conceptual level until the code level.

- There is a lack of standard that unify the basic concepts, vocabularies and transformation details.
- Most of proposals are concept-oriented and do not pay attention to the development perspectives, which play a key role in their adoption.
- There is duplication (reproduction) of research and development efforts instead of benefiting from existing approaches.

To address the above shortcomings, we assigned ourselves the next goals:

- To provide a single consolidated meta-model for each abstraction level presented in the Cameleon framework. These meta-models should strongly rely on existing proposals which presents common relevant concepts.
- To establish mapping between these meta-models leading to automatically generate the final user interface.

As is mentioned before, this paper presents a partial solution which only focuses on the first part that generates the abstract user interface from a task model. The next section describes our proposal to deal with such goal.

3 A UNIFORM PROCESS TO DEVELOP ABSTRACT USER INTERFACES

In this section, we present the steps followed to build a uniformed abstract user interface generation process. Our proposal consists of three major steps: consolidation of relevant common concepts and vocabularies required to design a task model into one single meta-model called uniformed task meta-model, consolidation of relevant common concepts and vocabularies required to design an abstract user interface model into one single meta-model called uniformed abstract user interface meta-model, proposing transformation rules enabling the generation of an abstract user interface from a task model.

Note that while proposing our meta-models some crucial concepts are implicitly taken into consideration, especially:

- Redundant information/concepts with different definitions/terminologies are omitted via a syntactical uniform process associating a uniformed label (usually a terminology which is the largely used/the most known in that context).
- Building the meta-models with a development-oriented perspective and thus facilitating their implementation and consequently their adoption in an industrial environment.

3.1 Uniformed Task Model

Task models are shown to be useful for designing interactive software applications. They describe how activities can be performed to reach the users goals when interacting with the application considered. After analyzing several related works, we opted for the adaption of the ConcurTaskTree (CTT) to represent user’s tasks along with their logical and temporal ordering. Some issues are kept in mind during the adaptation process as we consider them crucial for a task meta-model and may increase its adoption whether in the academic or industrial environment. These issues pertain especially to:

- Task hierarchy/decomposition; usually represented as a tree allowing the distinction between abstract tasks and concrete/elementary tasks and shows constraint for grouping related tasks in the interface.
- Relationship among tasks showing constraint for placing interaction objects.
- Domain-oriented concepts as they are required by tasks to be performed. Domain concepts are encapsulated in the task model for development reasons. We argue that reducing inputs and steps during the development process is likely to reduce complexity of the process.

A task model is therefore composed of tasks and relationships (Fig. 1). Tasks are described with a name, a description and a category. Task description represents domain-oriented concepts required by the task to be performed. Task category may be: users, interactive, system or abstract. User tasks are notably useful to describe tasks which are entirely performed by the user requiring only an internal cognitive activity without interaction with the system (e.g. selecting a strategy to solve a problem). An interactive task involves an active interaction of the user with the system (e.g., selecting a value, browsing a collection of items). A system task is an action that is performed by the application itself (e.g., check a credit card number). An abstract task is a task which requires complex actions, and their performance does not completely fall into one of the three previous cases.

To refine the expression of the nature of leaf tasks, two main attributes are considered: *UserAction* and *TaskItem*. Note that this expression is relied on the taxonomy introduced by (Constantine, 2003) to qualify a UI in terms of abstract actions it supports. The *UserAction* indicates a user action required for performing the task and the *TaskItem* refers to the type of object on which the action is operated. The derivation of interaction objects supposed to support a task is usually carried out by combining these two dimensions.

Task relationships allow specifying a temporal relationship between sibling tasks of a task tree. LOTOS operators are used here as they have been applied to task modelling in (Paterno et al., 1997). Note that, the hierarchical structure of the task model is supported using the *SubTask* composition. In addition, composition is used sometimes for some development reasons even the relationship can be simply modelled using an association (e.g. source and target of a relationship).

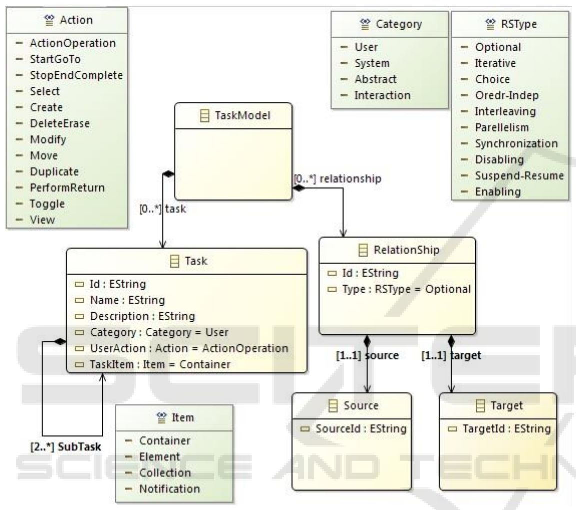

Figure 1: Task Meta-model.

3.2 Uniformed Abstract User Interface Model

An Abstract User Interface (AUI) model represents a canonical expression of the renderings and manipulation of the domain concepts and functions in a way that is as independent as possible from modalities and computing platform specificities. It is populated by *abstract interaction objects* (AIO) and *abstract relationship* (Fig. 2). AIO represents an abstraction of widgets found in most of the popular toolkit (e.g. windows, buttons, panels, etc.). It can be classified into two main types: *abstract containers* (AC) and *abstract individual components* (AIC).

AC (sometimes called presentation unit) is an entity allowing a logical grouping of other abstract containers or abstract individual components. It is said to support the representation of a set of logically/semantically connected tasks. It may be transformed, at the concrete level, into graphical container like windows, dialog boxes, etc.

AIC is an abstraction of an interaction object populating an abstract container independently of the modality in which it will be rendered in the concrete level. An AIC assumes at least one basic system interaction function described as facet in the UI. According to (Limbourg et al., 2001), four main facets can be identified: *Input*, *Output*, *Control* and *Navigation*.

The *UserAction* attribute of an AIC enables the specification of the type of action an AIC allows to perform. The *TaskItem* attribute characterizes the item manipulated by the AIC. The values of *UserAction* and *TaskItem* can be inherited from the *UserAction* and *TaskItem* attributes defined in the Task Model, respectively.

Abstract relationships are relationships that can be established between abstract interaction objects of all kinds. They are couples of *Source* and *Target*. *Abstract Adjacency* and *Spatio-temporal* are among the most common type of abstract relationships presented in the literature. Adjacency specifies an adjacency constraint between two AIOs. As for Spatiotemporal relation, two basic relations are considered in this paper: *sequential* and *simultaneous*.

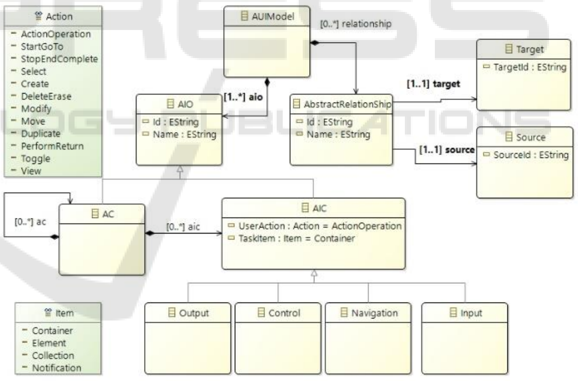

Figure 2: Abstract User Interface Meta-model.

3.3 Defining Transformation Rules

This section presents the process for automatic obtaining the Abstract UI from the task model. It is a recursive process which analyze the task tree in a top-down way. We opted for the Depth-first search (DFS) algorithm to implement this derivation process. This later may involve the following steps:

- Creation of Abstract Container: consists of the definition of the abstract UI structure via the definition of abstract container. This step will be applied for each parent node in the task tree (task which is split into several sub-tasks). The transformation rule to be executed is the following:

- **Rule 1:** for each parent note, create an abstract container. If the parent node is also a child of another parent node, link them with a containment relationship.

- Creation of Abstract Individual Component: consists of finding the adequate abstract individual component type to support one or several user's task. *UserAction* and *TaskItem* are notably important information to perform such adequate selection (see Algorithm 1). Rule 2 and 3 are examples of transformation rules that can be applied to enable the creation of an abstract individual component.
  - **Rule 2:** for each leaf task from the System type with a View UserAction, create an abstract individual component from the Output type. Link the abstract individual component and the abstract container corresponding to its parent node in the task tree by a containment relationship.
  - **Rule 3:** for each leaf task from the Interactive type with a Select UserAction, create an abstract individual component from the Input type. Link the abstract individual component and the abstract container corresponding to its parent node in the task tree by a containment relationship.
- Identification of spatio-temporal relationship: the spatiotemporal relationship defined between tasks can be respected in the abstract UI specification. This step consists of defining the spatio-temporal arrangement between elements of the abstract UI. Note that two level of arrangement are identified: 1) intra-container level concerns the arrangement of abstract individual component within the same abstract container, and 2) inter-container level concerns the definition of navigational structure among abstract container. The interest is focused in this paper on the second one. Rule 4 shows an example of a" Sequential" spatio-temporal arrangement between two abstract containers.
  - **Rule 4:** for each couple of abstract container connected by a relationship from the *Sequential* type, create a relationship between them "*Sequential".*

Algorithm 1: Task Tree Traversing.

```
for each task in task tree do
   if task has sub tasks then
   Create an AbstractContainer
   else
   if task is System then
         if UserAction is View then
         Create Abstract component
         from the Output type
         end if
         if UserAction is StartGoTo
         or StopEndComplete or
         PerformReturn then
         if TaskItem is Operation
         then
               Create Abstract
               component from the
               Control type
         end if
         if TaskItem is Container
         then
               Create Abstract
               component from the
               Navigation type
         end if
         end if
      end if
   if task is Interactive then
      if UserAction in Select, Create,
      Delete, Modify, Move, Duplicate
      then
      Create Abstract component from the
      Input type
      end if
   end if
   if task is User then do nothing
   end if
   end if
   end for
```

4 CONCLUSIONS

MBUID approach can contribute to the automatic generation of software process. One of the reasons that disallow the MBUID to have the expected success is the lack of standards and norms about concepts and terminologies used to design user interfaces. The objective of this paper is to elaborate a design framework aiming to unify concepts and terminologies required for user interfaces design. In this respect, a part of the intended proposition is presented allowing the generation of an abstract user interface from a task model. A uniform meta-model is presented for each level of abstraction allowing the design of the user interface. In addition, a set of transformation rules are presented showing the mapping process between the task model and abstract UI model. In the expected framework, a user interface undergoes three transformations starting from a task to automatically obtain the final user interface. Future works includes the full implementation of this step from the development process (T&C to Abstract UI) and the consideration of the other levels of abstraction

(CUI) with regard to the unifying of concepts and terminologies.

REFERENCES

Molina, A. I., Giraldo, W. J., Gallardo, J., Redondo, M. A., Ortega, M., and Garcia, G. "Ciat-gui: A mde-compliant environment for developing graphical user interfaces of information systems," *Adv. Eng. Softw.*, vol. 52, pp. 10–29, Oct. 2012.

Meixner, G., Patern, F., and Vanderdonckt, J. "Past, present, and future of model-based user interface development." icom, vol. 10, no. 3, pp. 2–11, 2011.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. "A unifying reference framework for multi-target user interfaces," *Interacting with Computers*, vol. 15, no. 3, pp. 289–308, jun 2003.

Vanderdonckt, J. and Bodart, F. "Encapsulating knowledge for intelligent automatic interaction objects selection," *in Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems, ser. CHI '93*. New York, NY, USA: ACM, 1993, pp. 424–429.

Akiki, P. A., Bandara, A. K., and Yu, Y. "Engineering adaptive model driven user interfaces," *IEEE Transactions on Software Engineering*, vol. 42, no. 12, pp. 1118–1147, Dec 2016.

Molina, A. I., Giraldo, W. J., Ortega, M., Redondo, M. A., and Collazos, C. A. "Model-driven development of interactive groupware systems: Integration into the software development process," *Sci. Comput. Program.*, vol. 89, pp. 320–349, 2014.

Tran, V.., Vanderdonckt, J., Tesoriero, R., and Beuvens, F. "Systematic generation of abstract user interfaces*," in Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ser. EICS '12. New York, NY, USA: ACM, 2012, pp. 101–110.

Limbourg, Q., Pribeanu, C., and Vanderdonckt, J. "Towards uniformed task models in a model-based approach," in *Interactive Systems: Design, Specification, and Verification*, C. Johnson, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 164–182.

MBUI. Model-based user interfaces (mbui) working group. [Online]. Available: https://www.w3.org/TR/

Constantine, L. L. "Canonical abstract prototypes for abstract visual and interaction design," *in Interactive Systems. Design, Specification, and Verification*, J. A. Jorge, N. Jardim Nunes, and J. Falc˜ao e Cunha, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 1–15.

Paterno, F., Mancini, S. and Meniconi, C., ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. Boston, MA: Springer US, 1997, pp. 362–369.