Towards an Integrated Model for Functional and User Interface Requirements

Rabeb Mizouni[1], Daniel Sinnig[2], Ferhat Khendek[3],
[1] College of Information Technology, UAE University, Al-Ain, UAE.
mizouni@uaeu.ac.ae
[2] Faculty of CS and Elec. Engineering, University of Rostock, Germany.
dasin@informatik.uni-rostock.de
[3] Dept. of Electrical & Computer Eng., Concordia University, Montreal, Canada.
khendek@encs.concordia.ca

Abstract: Despite the widespread adoption of UML as a standard for modeling software systems, it does not provide adequate support for specifying User Interface (UI) requirements. It has become a common practice to separately use UML use cases for specifying functional requirements and task models for modeling UI requirements. The lack of integration of these two related models is likely to introduce redundancies and inconsistencies into the software development process. In this paper, we propose an integrated model, consisting of use case and task models, for capturing functional and UI requirements. Both artifacts are used in a complementary manner and are formally related through so-called *Anchors*. Anchors are use case steps that require further elaboration with UI-specific interactions. These interactions are explicitly captured in associated task models. The formal semantics of the integrated model is given with finite state automata.

Keywords: Functional Requirements, UML Use Cases, User Interface Requirements, Task Models, Integrated Requirements Model, Finite State Automata.

1 Introduction

UML has become the de-facto standard for software systems modeling. However, UML's support for User Interface (UI) development is deemed insufficient [1]. While UML diagrams are well suited for object-oriented analysis and design, the HCI community argues that a set of specialized models is needed to effectively specify users' characteristics and tasks, UI dialogue structures and layouts.

This divergence has been addressed by many researchers. Most attempts either define extensions for UML to capture HCI related information [2, 3] or, conversely, extend HCI models to cope with object-oriented features [4, 5]. An effective integration, however, is not simply a matter of expressiveness and the ability to convert or embed a model into another one. Instead, as Paternò [1] points out, specialized notations should be used in a complementary manner to efficiently support software engineers and UI designers in their work.

In this paper, we define an integrated model for capturing functional and UI requirements. It is composed of two heterogeneous, yet interrelated parts: *UML use cases* and *HCI task models.* Use cases are the medium of choice for capturing functional requirements whereas task models are commonly used to specify the

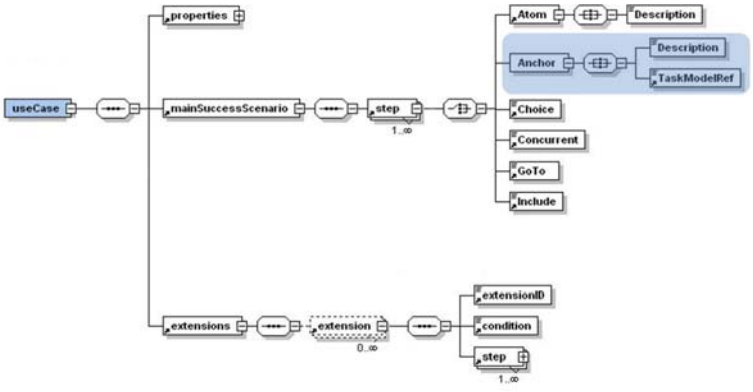

Fig. 1. Use Case Model Syntax with Anchor

detailed user interactions with the system. Within our integrated model, use cases and task models are used according to their intended purposes establishing clear separation of concerns.

The research reported here builds upon our earlier work [6, 7] where we described a two-phase integrated development methodology for use cases and task models. In the *first* phase, an initial coarse grained use case model is developed, which, without delving into details, documents the primary interactions that actors will perform with the system in a step-by-step format. Additionally, for each use case, the software engineers identify a set of use case steps that require further elaboration with UI details. These steps are called *anchors*. In the *second* phase, each anchor is associated with a corresponding task model capturing UI-specific interactions. Concurrently, the coarse-grained use case model is further refined by taking into account alternative and failure cases that thus far have been considered only marginally.

In this paper we focus on the definition of an integrated model for functional and UI requirements to support such a methodology, including its syntax and semantics. The latter is defined by providing a formal mapping to the semantic domain of finite state automata.

2 Syntax of the Integrated Functional and UI Requirements Model

In this section we define the syntax of our integrated model for functional and UI requirements. As aforementioned, the model consists of two heterogeneous parts, a use case model and a set of task models, interlinked by a set of anchor points. Each individual use case corresponds to the structure portrayed in Fig. 1. The main success scenario as well as each extension consist of a sequence of use-case steps, which can be of six different kinds. *Atomic* steps are performed either by the system or a secondary actor. They contain a textual description, but do not consist of any sub-steps. *Anchors* are also atomic, but are performed by the primary actor and as such are related to the user interface. Anchor steps additionally contain a reference to a refining (UI-specific) task model. *Choice* steps provide the primary actor with the choice between several interactions. Each such interaction is (in turn) defined by a

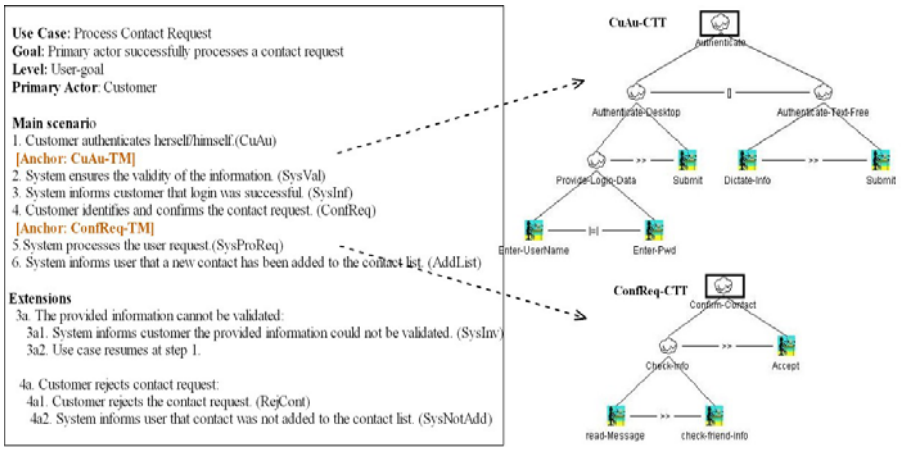

Fig. 2. Integrated Functional and UI Requirements Model of the "Process Contact Request" Use Case

sequence of steps. *Concurrent* steps define a set of steps which may be performed in any order by the primary actor. *Goto* steps denote jumps to steps within the same use case. *Include* steps denote invocations of sub-use cases.

To illustrate our approach, let us consider a "Process Contact Request" use case. It depicts the interactions involved in processing contact requests, as it is typical in social networks such as LinkedIn and Facebook. The main success scenario describes the situation in which the primary actor directly accomplishes his goal of confirming a contact request. We also define two extensions to specify alternative scenarios, which occur when the primary actor fails to authenticate himself or refuses a contact request, respectively. The textual description of the use case is shown on the left hand side of Fig. 2.

The use case contains two UI-related steps 1 ("Authentication") and 4 ("Identification of Contact Request") which are defined as anchors and as such are related to refining task models. Both steps do not detail how the step-goals are achieved. These interactions are UI-specific and are captured in the corresponding task models. For example, the authentication step (CuAu) may require that the user enters his/her name and password in any order (Desktop UI), or that the user dictates his/her login information (Text-Free Voice UI). Both possibilities are expressed by the binary choice operator ([]) in the corresponding task model (CuAu-TM). In a similar manner, use case step 4 (ConfReq) is associated with a task model (ConfReq-TM), specifying UI interactions for confirming a contact request.

3 Semantics of the Integrated Functional and UI Requirements Model

This section defines a formal semantics for our integrated model. We start by defining the well-known semantic domain of finite state automata. We then portray how the use case and task model parts of the integrated model are mapped separately into the

semantic domain. Finally, we define a merging procedure that integrates the various individual semantic representations into a common behavioral model.

3.1 Semantic Domain

The semantics for our integrated model is given by a mapping to a finite state automaton.

Definition 1 (Finite state automaton). An automaton is defined as a 5-tuple(S,s^0,S^f,L,E) where S is the set of states, s^0 is the initial state, S^f is the set of final states, L is the set of labels, and $E \subseteq S \times L \times S$ is the set of transitions.

Definition 2 (Trace). A trace of an automaton A is a sequence of transitions $e=q_0.q_1.q_2.\ldots q_{n-1}$ where $q_0=(s_0,l_0,s_1) \in E$ such that $s_0=s^0$, $\forall i, 1 \leq i<n-1$ $q_i=(s_i,l_i,s_{i+1}) \in E$, and , $q_{n-1}=(s_{n-1},l_{n-1},s_n) \in E$ where $s_n \in S^f$

Informally, a trace is a word of the language accepted by the finite state automaton when it starts from its initial state and ends in one of its final states for the trace. In what follows, we use operational semantics for our definitions. Equations of the following form $\frac{a\,;b}{c}(Cond)$ denote that *a AND b IMPLY c*. *Cond* is the condition for the applicability of the rule.

3.2 Semantics for Use Cases and CTT Task Models

This section outlines the separate mappings of the use case and task model into the semantic domain. For the sake of conciseness, only a high-level overview will be given while the full details can be found in [8].

The semantic mapping from a use case model into an automaton is defined in a bottom-up manner, starting with the mapping of individual use case steps. Each of the six kinds of use case steps enumerated in Fig.1 has its own specific mapping to an FSM. *Atomic* steps and *Anchor* steps map to elementary FSMs consisting of only an initial state and a set of final states, connected by a transition that represents the use case step. A *Choice* step maps to a composite FSM consisting of the initial states of each choice's FSM. A *Concurrent* step is the product machine of its constituent FSMs. *Goto* steps map to an FSM with a single state defined to be equivalent to the initial state of the FSM representing the target of the jump. The complement FSM of an *Include* step consists of two states: one identified with the initial state of the FSM of the main success scenario of the invoked sub-use case, and the other identified with all final states of the sub-use case's FSM.

Now that individual use case steps can be formally represented by automata, we can link arbitrary sequences of steps using *sequential composition*, by unifying the final states of the first operand with the initial state of the second one. In the next step, we map the main success scenario and each extension of the use case to a set of automata, each being the result of the sequential composition of the automata representing the individual use case steps. Finally, the entire use case is mapped into an automaton, by merging the automata representing the main success scenario and all its extensions.

Similar to the semantic mapping of use cases, the mapping of CTT task models to automata is performed in a bottom-up manner. Each atomic task is mapped into an atomic automaton. Composite tasks are represented by more complex automata, which result from the composition of the automata representing sub-tasks. We have defined the following composition operations: *sequential composition* (•), *choice composition* (#), *parallel composition* (||), and *iterative composition* (∗). The full details of the mappings are given in [8].

3.3 Automaton of the Integrated Model

In this section, we elaborate how the individual use case and task model automata are merged into a single automaton, representing the behavioral semantics of our integrated requirements model.

Intuitively, the behavior of the integrated requirements model can be summarized as follows: At first, the integrated model adopts the behavior of the use case model up until an *Anchor* step is encountered. At this point, the integrated model adopts the behavior of the associated CTT model depicting how the primary actor may accomplish the step-goal using a particular UI. Thereafter, the integrated model again resumes with the behavior of the use case model. This alternating continues until the scenario comes to an end.

The behavioral merge of finite state automata has been addressed in many research projects [6, 9-11]. Since in our integrated model use cases and task models are utilized in a complementary –non-overlapping– manner we choose one of the existing *explicit* automata composition techniques [6, 10, 11] to merge the respective use case and task model automata. Similar to our work presented in [6], the merge of use case and task model automata is based on imperative expressions. Each expression specifies (1) the use case and the CTT automata to be merged, (2) the anchor where the merge is performed, and (3) a *Refine* operator that specifies how the actual merge is performed. The evaluation of the expression yields a new automaton where the use case transition representing the anchor step has been replaced by the corresponding CTT automaton. We define *Refine* operator semantics next.

Let $A=(S,s^0,S^f,L,E)$ be an automaton and let $tr(A)=\{e \mid e \text{ is a trace of } A\}$ be its set of traces. We define the set $tr(A, ep)$ to be the set of traces of A passing through the anchor ep as: $tr(A,ep)=\{e \in tr(A), e= q_0.q_1.q_2.\ldots.q_{n-1}, \forall 0\leq i\leq n-1\ q_i\in E \mid \exists\ q_i = ep\}$. It represents the set of traces where ep appears as a transition in the trace. Additionally, we define $Pref(A,ep)$ (respectively postfixes $Post(A,ep)$) as the set of prefixes (respectively postfixes) of the traces of the automaton A passing through ep. More formally, let $e=q_0.q_1.q_2.\ldots.q_{n-1}$ be a trace of automaton A. $u= q_0.q_1.\ldots.q_{i-1} \in Pref(A, ep)$ *if* $(q_i = ep)$ (respectively, $r=q_{i+1}.\ldots.q_{n-1} \in Post(A, ep)$ *if* $(q_i = ep)\}$. Consequently, a trace $e \in tr(A,ep)$ can be written as: $e=u.ep.r$ where $u\in Pref(A,ep)$ and $r \in Post(A,ep)$.

Definition 4 (Refine Operator): Let $A=(S_1,s^0_1,S^f_1,L_1,E_1)$ and $B=(S_2,s^0_2,S^f_2,L_2,E_2)$ two automata, and $C=(S_3,s^0_3,S^f_3,L_3,E_3)$ be the resulting automaton by applying the *Refine* operator at the anchor $t=(s,a,s')$. Furthermore, let $tr(A)$ and $tr(B)$ be the traces

of automata *A* and *B*, respectively. Then, the set of traces *tr(C)* of the resulting automaton *C* is constructed using the following rules:

$$(1)\ \frac{(e \in tr(A) / tr(A,t))}{e \in tr(C)}$$

$$(2)\ \frac{(e \in tr(A,t)); (\exists u, r \mid e = u.t.r \ \ where\ u \in \Pr ef(A,t), r \in Post(A,t)); e_b \in tr(B))}{u.e_b.r \in tr(C)}$$

Equation (1) shows that all traces of *A* not passing through the anchor point are traces of the automaton *C*. Equation (2) shows that for all traces of *A* passing through *t*, the transition *t* is replaced by the traces of *B*.

The construction the final merged automaton is an iterative process, where the resulting automaton from a composition is used as an input to a subsequent composition until a fixpoint is reached (i.e., all anchors have been replaced by respective task model automata). At the end of the composition, the derived automaton is the semantic representation of the integrated requirements model. Fig.3 portrays the various automata involved in our "Process Contact Request" example. The automaton representing the use case is given in Fig. 3 (a). The task model automata representing the refinements of the "authentication step" and the "confirm contact" step are given in Fig. 3 (b) and Fig. 3 (c). Finally, Fig. 3 (d) illustrates the resulting automaton representing the integrated model.

4 Related Work

Since UML was developed with little attention to UI related issues, several proposals have been put forward to close this gap. Most of them fall into one of the following categories: (1) Extensions to UML for the purpose of capturing HCI-related information [2, 3], (2) extensions to HCI models for capturing object-oriented features [4, 5] and (3) development methods promoting the integration of HCI and OO models [12].

Paternò [1] proposes a method for integrating use case diagrams and task models. Use cases denote core functionalities offered by the system which are refined by a set of task models. However, the scenario descriptions entailed in each use case are not taken into account. Another approach that falls under the first category is presented by Noberga *et al*. [2]. Motivated by the fact that the current UML standard provides insufficient support for modeling interactive systems, a mapping from CTT task models to UML activity diagrams is proposed. The mapping is complemented by an extension of UML with high-level syntactic constructs related to task modeling.

Da Silva and Paton [5] propose UML*i* as a modeling language for interactive systems. UML*i* extends UML with UI diagrams for describing abstract interaction objects. According to their eight-step methodology, use cases are employed to define high-level functionalities which are further refined by a set of user tasks captured in extended UML activity diagrams. A set of logical links, placed between the various use cases and the activity diagrams, establishes traceability between UI details and the corresponding functional requirements.

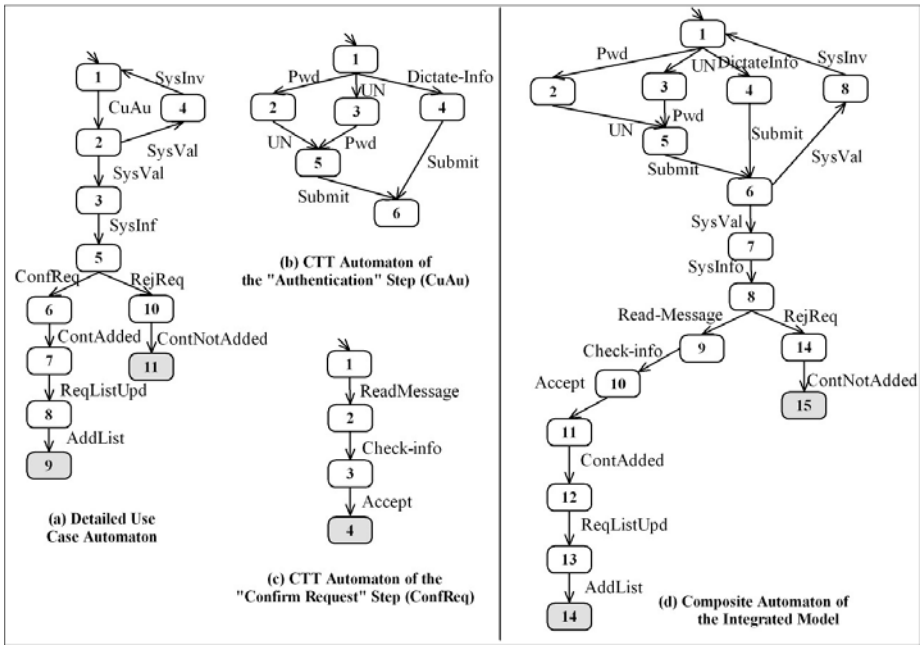

Fig. 3. Composition Example

Rosson [13] proposes a scenario-based approach to object-oriented analysis and design. In order to integrate usability concerns with functional modeling, a system is modeled by a set of *instance scenarios*. In a bottom-up approach the various scenarios are processed and serve as a basis for the creation of the object model. Nunes and Conha [4] point out that UML provides inadequate support for modeling architectural concerns of interactive systems and propose their Wisdom framework to fill this gap. While mainly based on existing UML models, Wisdom introduces a CTT-like notation to capture the dialogue between users and the application.

5 Conclusion

In order to overcome the insufficient support for UI modeling in UML, we have proposed an integrated model to capture functional and UI requirements. This integrated model is the outcome of a larger undertaking, first discussed in [7], that investigates methods for efficient collaboration between software engineers and UI designers while preserving clear separation of concerns. The integrated model is comprised of two well-established models − UML use cases and CTT task models – interrelated through a set of Anchors. We have defined a formal syntax and semantics for the integrated model. The latter is given in terms of a finite state automaton.

As future work, we plan to carry out comprehensive case studies and to apply our approach and notation to industrial-strength projects. We are currently developing tool support for authoring and validating the integrated model. We envision that our

tool will support the reuse of task model specifications (either within the same project or among different projects). In many cases, the interactions specified by task models are independent from the application domain and consequently can be reused across projects. Other future avenues are related to the extension of the integrated model to encompass other UI-related artifacts such as user and dialogue models and the generation of integrated test cases.

References

1. Paternò, F. and C. Santoro, *Support for Reasoning about Interactive Systems through Human-Computer Interaction Designers' Representations.* Comput. J., 2003. **46**(4): p. 340-357.
2. Nobrega, L., N.J. Nunes, and H. Coelho, *Mapping ConcurTaskTrees into UML 2.0.* Gilroy, S.W., Harrison, M.D. (eds) Interactive System, 2006. **3941**.
3. Bastide, R.e., *An Integration of Task and Use Case Metamodels.* HCI International, San Diego, CA, USA, 19/07/09-24/07/09, 2009.
4. Nunes, N.J. and J.o.F. o e Cunha, *Towards a UML profile for interaction design: the Wisdom approach.* 2000.
5. de Paula, M.i.G., B.S. da Silva, and S.D.J. Barbosa, *Using an interaction model as a resource for communication in design.* CHI '05: CHI '05 extended abstracts on Human factors in computing systems, 2005: p. 1713-1716.
6. Mizouni, R., et al., *Merging partial system behaviors: composition of use-case automata.* IET Software, 2007. **1**(4): p. 143-160.
7. Daniel, S., M. Rabeb, and K. Ferhat, *Bridging the gap: empowering use cases with task models*, in *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*. 2010, ACM: Berlin, Germany.
8. Sinnig, D., *Use Case and Task Models: Formal Unification and Integrated Development Methodology*. 2008, Department of Computer Science and Software Engineering, Concordia University, Montreal.
9. Chechik, M., et al., *Partial Behavioral Models for Requirements and Early Design.* MMOSS, 2006. **06351**.
10. S.Leue, L. Mehrmann, and M. Rezai, *Synthesizing ROOM Models from Message Sequence Chart Specifications.* Technical Report 98-06, ECE Dept., University of Waterloo, Canada, october 1998.
11. Uchitel, S., J. Kramer, and J. Magee, *Behavior Model Elaboration using Partial Labeled Transition Systems* ESEC/FSE 2003, 2003.
12. Lu, S., et al., *Generating UML Diagrams from Task Models.* 2003.
13. Rosson, M.B., *Integrating development of task and object models.* Commun. ACM, 1999. **42**(1): p. 49-56.