An Integration of Task and Use-Case Meta-models

Rémi Bastide

IRIT – Université de Toulouse,
ISIS – CUFR J.F. Champollion, Castres, France
Remi.Bastide@irit.fr

Abstract. Although task modeling is a recommended practice in the Human-Computer Interaction community, its acceptance in the Software Engineering community is slow. One likely reason for this is the weak integration between task models and other models commonly used in Software Engineering, notably the set of models promoted by the mainstream UML method. To overcome this problem, we propose to integrate the CTT model of user tasks into the UML, at the meta-model level. CTT task models are used to provide an unambiguous model of the behavior of UML use-cases. By so doing, we also bring the benefit of hierarchical decomposition of use-cases ("extend" and "include" relationships) to CTT. In our approach, CTT tasks also explicitly operate on a UML domain model, by using OCL expressions over a UML object model to express the pre- and post-conditions of tasks.

1 Introduction

In the current Software Engineering practice, use-cases are routinely used during the early phases of software development, namely requirements gathering. Use-cases are arguably the less formalized of all UML notations. Rather than a hindrance, this is to be considered as an advantage: the main point of use-case modeling is to reach a common understanding of the problem between the various stakeholders of the system under development, and especially between the customer (who holds the knowledge of the business domain) and the software development team (who has the know-how of the software development process). Noted methodologists [2] argue that writing good use-cases is essentially a literary piece of work, and that a natural language description of use-cases is a good way to form consensus and mutual understanding between the stakeholders regarding *what* has to be done, regardless of *how* it has to be done.

A delicate point comes with the need to relate an informal, natural language description of use cases to the increasingly formal notations used in the UML, for instance class diagrams, behavioral models such as StateCharts, etc, eventually leading to a satisfactory implementation. UML is notoriously vague and non-prescriptive with regards to the precise way to describe the behavior of use-cases. Some authors stick to a detailed natural language scenario, others prefer a partitioned narration, and others use UML sequence diagrams to describe the information exchanges between the use-case actor and the system under design. We contend that task models (in our case

J.A. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2009, LNCS 5610, pp. 579–586, 2009.

CTT models [8]) offer several advantages over the latter two, notably due to the richness of the temporal operators available. This is increasingly important, since modern user interfaces (direct-manipulation, multi-modal...) depart from the old-fashioned conversational, question-answer style, and are almost impossible to model with sequence diagrams.

It is also a routine practice to develop an analysis model of the business objects of the system under design (the so-called "domain model") early on in the development process, in order to precisely identify the business objects, their structure and their mutual relationships. This domain modeling is performed using UML class diagrams, leaving out premature implementation-related considerations.

The main point of this paper is to promote CTT task models as the behavioral language for use cases. To this end, we first introduce our view of the design process which is expected. We then show how the metamodel of CTT can be tightly integrated into the UML metamodel of use-case diagrams, so that the notion of *extend* and *include* relationships become meaningful for CTT task models.

2 Design Process

For the sake of efficiency, formal modeling work has to be guided by strong methodological, process-oriented guidelines. A design process defines in which order the various artifacts have to be produced during the software lifecycle, defines the expects contents of these artifacts, and what information is needed as an input and produced as an output of the various modeling and design activities.

The work presented here deals mainly with the initial phases of the process, namely requirements engineering and preliminary design.

- The goal of the *requirements engineering* phase if to form a consensus between the stakeholders (mainly the customer and the analysis team) regarding *what* the problem actually is, and what has to be developed in order to solve the problem. The main outcome of this phase is a common understanding between the customer and the development team of the *problem domain*: no work on the solution domain should be performed at this phase.
- Work on the *solution domain* begins at the *preliminary design* phase: this is where the first decisions on software architecture are made, and where the best practices of interaction design (in particular iterative prototyping with increasing fidelity level) should be used.

Of course, we do not recommend a strict separation between these two phases: it is quite common that work performed at the preliminary design phase uncovers new unforeseen insights on the requirements, and that some iteration has to be performed between these two phases. Although iteration is frequent between these two phases, it should always remain clear to the various actors whether they are working on the problem domain (i.e. the requirements) or on the solution domain (i.e. the design).

Our claim is that task modeling is especially useful during the requirements engineering phase, and that it complements nicely the domain models and use-case

models that are developed during this phase. At this stage, class models are used to provide an analysis-level model of the domain (they formalize the vocabulary of the business domain), while use-cases and use-case diagrams are used to provide a user-oriented view of the system functionality. The natural language scenarios that are associated with use-cases are essential in easing the construction of a common understanding of the problem between the stakeholders, since they are written in the vocabulary of the business and can be understood and validated by the customer.

Our view that task models are essentially a requirement analysis tool contradicts several authors, who recommend using task models at the design phase, for instance to drive the generation of dialogue [6] or abstract interface models. In our approach, requirement task models necessarily remain at a rather abstract level, since at this stage the user interface is not (and should not be) yet under design. It follows that requirement task models should not mention any user-interface specifics: rather, the task models will drive the user-centered design of the UI that will follow in the subsequent phases, where the user interface specialist will strive to design an interface that is best suited to the user task, while taking into account the limitations inherent to the target platform for the interactive system. We do not believe that (except maybe in very stereotyped situations, such as business form-filling applications) a satisfactory user interface can be automatically generated from a task model. Rather, in our view, the task model can be used as a test case for the user interface that will be designed using user-centered techniques such as incremental low-fidelity prototyping.

To allow for the smooth integration of task models in the software design lifecycle, we propose to integrate task models and use-cases at the meta-model level [5, 14], thus opening the way for efficient use of Model-Driven Engineering (MDE) techniques such as model weaving and model transformation.

The process we advocate is inspired by the "essential use-cases" work proposed by Constantine and Lockwood [4] and the work in [13]. In particular, since use-cases are meant the be an input to interaction design, they should be devoid of any specific reference to user interface, otherwise it would be a premature commitment to a user interface design, before this design has been presented and validated by users through low-fidelity prototyping.

We propose that CTT task models should serve as the behavioral language for use-cases. In this usage of task modeling, task models are meant to provide an abstract view of the user's activity, exploring their goals as well as the temporal and causal structure of their interactions with the system. Task models are thus the formal counterpart of the natural language, narrative descriptions of scenarios that is routinely associated with use-cases, and that are still quite useful: natural languages scenarios are ideal to communicate and form consensus with the customer, and can be developed and validated with the customer during brainstorming sessions. Task models, on the other hand, are useful to communicate with the design team, since they convey a precise semantics of the dynamics of human-computer interaction that has to be supported by the software to be produced.

Fig. 1 illustrates our view of the early stages of the design process, highlighting the strong bonds between use-cases, domain model and task models that are the main outcomes of the requirements analysis phase.

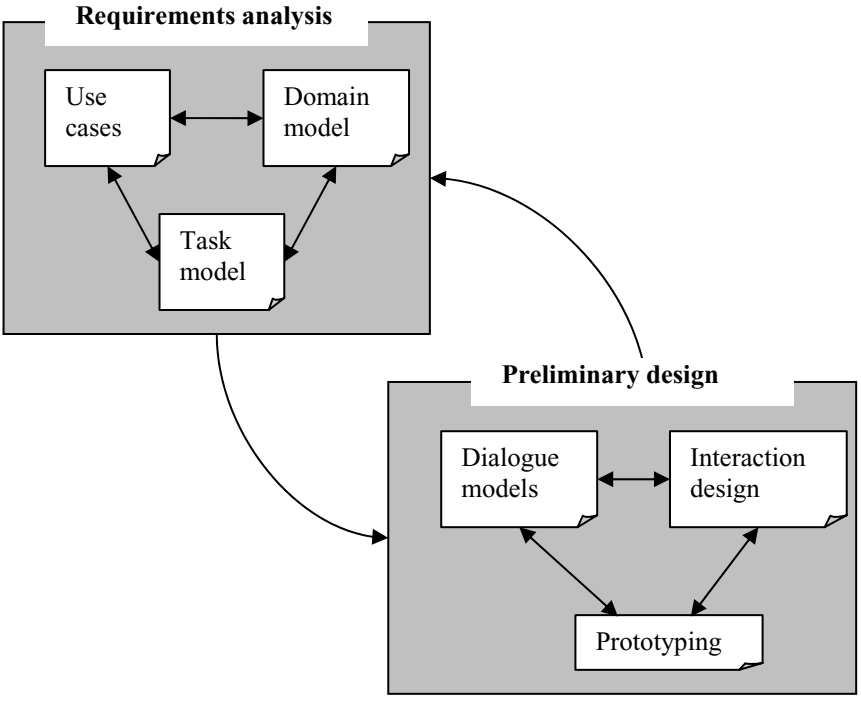

Fig. 1. First stages of the design process

3 Related Work

The need to bridge the gap between the current practices of Software Engineering (centered on UML diagrams) and user-centered design (including task analysis and modeling) has been stressed by numerous authors. A remarkable variety of solutions to this problem has been proposed. The very father of the CTT notation [12] has identified the main trends of work in this field:

- Representing CTT by an existing notation: Nobrega et al. [10], for instance, provide semantics of the temporal operators of CTT in terms of UML activity diagrams. Nunes et al. [9] use the extensions mechanisms provided by the UML (profiles, stereotypes) to represent the concepts of CTT in a UML framework.
- Developing automatic converters from UML to task models [6] (and back, we should add). It can be contended that, in the HCI literature, one can find proposals for generators from any kind of model to any other kind.
- Building a new UML for interactive systems "which can be obtained by explicitly inserting CTT in the set of available notations" [10]. This is the trend we follow in this paper, by integrating a metamodel of CTT inside the metamodel of UML itself.

Although we share the goals expressed in [10], our technical proposal is quite different with the one presented there.

- In the first place, we work formally at the metamodel level, whereas only a rough sketch of a solution was provided in [10]. We believe that explicit use of metamodels brings several fundamental advantages, including the opportunity to use existing MDE tools such as model transformation languages or model weavers to extend the potential use of models. We have demonstrated this advantage in previous work [1], by showing how the notions of human errors can be integrated in task diagrams through the use of error patterns and automatic model transformations.
- Furthermore, it appears that our proposal is almost an "inside-out" reversal of the approach in [10] : the authors proposed a path to transform a use-case diagram (also called a use-case map) into a CTT task model, that could be further refined. In the contrary, we propose to use CTT as a language to specify the behavior of use-cases.

4 Alignment with the UML Use-Case Metamodel

The metamodel of UML use-cases is given in Fig. 2. This is actually the metamodel of *use-case maps* (diagrams that show the relationships between the use cases for a system), since UML is non-prescriptive as to what a use case actually is, i.e. as to what the behavioral description of a use-case should be.

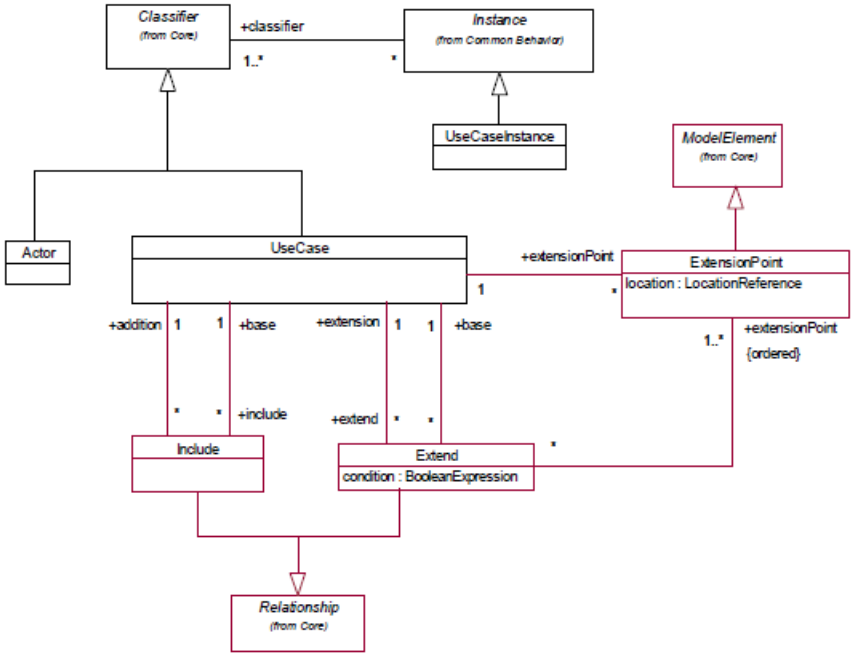

Fig. 2. The UML use-case metamodel (from [11])

There has been some picky debate amongst specialists over this very metamodel [15], several of its flaws have been pointed out, and better alternative metamodels have been proposed. Although we mainly agree with these criticisms, we have chosen to stick with the "official" metamodel, since our goal is to be as close as possible to the standard. It should also be noticed that the ill-defined notion of use-case specialization relationship, formerly available in the UML, has been removed in the current version of the standard.

Starting from this "official" metamodel of UML use-cases we want to cleanly integrate a metamodel of CTT, in order to express that a CTT task model is used to express the behavior of a use-case, and to show that "include" and "extend" relationships can be expressed over a CTT model.

The metamodel of CTT illustrated in Fig. 3 improves on the one we previously published in [1] in several ways:

- Our earlier metamodel used eCore [14] Ecore as the metamodeling language. The one presented here uses UML class diagrams for the same purpose, which allows us to cleanly express its relationships with other elements in the UML metamodel. For instance, it expresses that the notion of Actor in CTT is identical with the same notion in UML use cases. In turn, this enriches CTT with the features available for UML actors (for instance, one can design a specialization hierarchy of actors with increasing responsibilities)
- It explicitly aligns CTT with UML use cases, bringing their structuring features ("include" and "extend" relationships) to CTT.

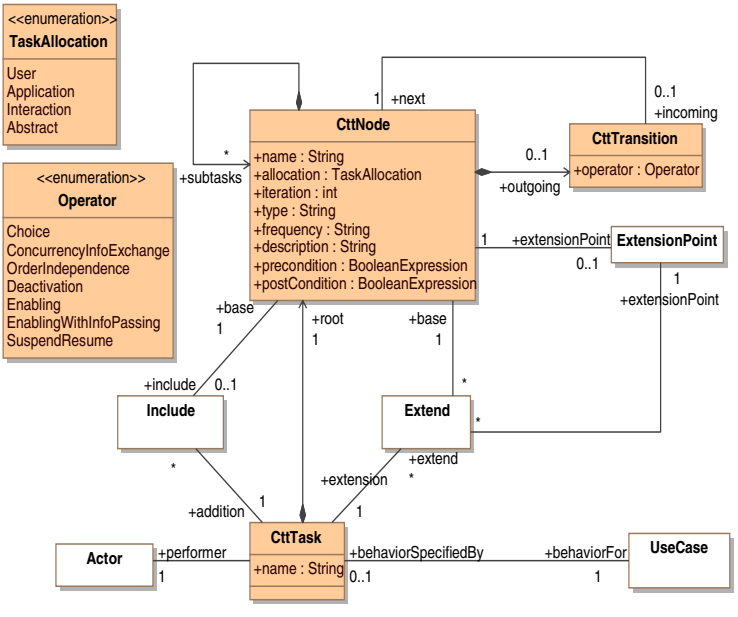

Fig. 3. A metamodel of CTT integrated in the metamodel of UML

In Fig. 3, the classes with a white background are imported from the UML metamodel, and should be related to the identical ones in Fig. 2. The classes with the filled background are specific to CTT. Basically, a CTT task model (*CttTask*) is a tree of nodes (*CttNode*) which can be related by transitions (*CttTransition*) that feature one of the CTT temporal operators (*Operator*).

The use-case metamodel of Fig. 2 states that a use-case can have several "extend" and "use" relationships (* cardinality). The cardinalities chosen in our metamodel of CTT in Fig.3 should be carefully considered:

- *Include* relationship: a *CttNode* has 0..1 include relationships, meaning that any CTT node can optionally include another *CttTask* (which in turn is a tree of *CttNodes*). This models a classical hierarchical decomposition, which makes it easy to reuse a task model in another one, by simply including it at the proper node. It is natural to allow for a maximum of one inclusion, since otherwise the temporal combination of the included *CttTasks* would be left undefined.
- *Extend* relationship: an extend relationship is ternary, relating a *base* to an *extension* through one *extensionPoint*. In our metamodel, a CttNode has an optional extensionPoint, meaning that it can be optionally extended. However, a *CttNode* can have several *extensions*, discriminated by `condition: BooleanExpression` in metaclass *Extend* (cf. Fig. 2).

It is noteworthy that the metamodel in Fig. 3 conveys the same information as the initial use-case metamodel, only more so. For instance, the set of *Include* relationships for a given use-case (which are actually the relationships appearing on use case maps) can be computed by exploiting the Include and Extend relationships of Fig. 3 recursively using the hierarchical composition relationship between *CTTNodes*.

The metamodel in Fig.3 also relates to the domain model, albeit implicitly: the *preConditions* an *postConditions* elements in *CttNode* are meant to be Boolean expressions expressed in OCL (Object Constraint Language) operating on a domain model defined by a UML class diagram. As OCL itself is not part of the UML metamodel, but defined in a separate, language-oriented specification the relationship between task and domain model is not apparent, but is nonetheless fundamental.

5 Conclusion

We have presented our view of a design process where task and use-case modeling are tightly integrated during the requirement engineering phase. CTT task models are used to provide an unambiguous description of use-case behavior, complementing natural language scenarios. An integration of CTT into the UML metamodel has also been presented, which opens the way to automatic processing of requirement models, to be use in subsequent phases of the design and implementation, for instance test sequence generation.

References

1. Bastide, R., Basnyat, S.: Error Patterns: Systematic Investigation of Deviations in Task Models. In: Coninx, K., Luyten, K., Schneider, K.A. (eds.) TAMODIA 2006. LNCS, vol. 4385, pp. 109–121. Springer, Heidelberg (2007)
2. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley Professional, Reading

3. Constantine, L., Campos, P.: Canonsketch and tasksketch: innovative modeling tools for usage-centered design. In: OOPSLA 2005: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pp. 162–163. ACM, New York (2005)
4. Constantine, L.L., Lockwood, L.A.D.: Constantine & lockwood, ltd. structure and style in use cases for user interface design
5. Limbourg, Q., Pribeanu, C., Vanderdonckt, J.: Towards Uniformed Task Models in a Model-Based Approach. In: Johnson, C. (ed.) DSV-IS 2001. LNCS, vol. 2220, pp. 164–182. Springer, Heidelberg (2001)
6. Luyten, K., Clerckx, T., Coninx, K., Vanderdonckt, J.: Derivation of a dialog model from a task model by activity chain extraction (2003)
7. Montero, F., López-Jaquero, V., Vanderdonckt, J., González, P., Lozano, M.D., Limbourg, Q.: Solving the mapping problem in user interface design by seamless integration in idealXML. In: Gilroy, S.W., Harrison, M.D. (eds.) DSV-IS 2005. LNCS, vol. 3941, pp. 13–15. Springer, Heidelberg (2006)
8. Mori, G., Paterno, F., Santoro, C.: Ctte: Support for developing and analyzing task models for interactive system design. IEEE Trans. Software Eng. 28(8), 797–813 (2002)
9. Jardim Nunes, N., Falcão e Cunha, J.: Towards a UML profile for interaction design: The wisdom approach. In: Evans, A., Kent, S., Selic, B. (eds.) UML 2000. LNCS, vol. 1939, pp. 101–116. Springer, Heidelberg (2000)
10. Nóbrega, L., Jardim Nunes, N., Coelho, H.: Mapping ConcurTaskTrees into UML 2.0. In: Gilroy, S.W., Harrison, M.D. (eds.) DSV-IS 2005. LNCS, vol. 3941, pp. 237–248. Springer, Heidelberg (2006)
11. Object Management Group: Unified Modeling Language (UML), version 2.0. Technical report, OMG (2005), `http://www.omg.org/technology/documents/formal/uml.htm`
12. Paternó, F.: Towards a UML for interactive systems. In: Nigay, L., Little, M.R. (eds.) EHCI 2001. LNCS, vol. 2254, pp. 7–18. Springer, Heidelberg (2001)
13. Rosson, M.B.: Integrating development of task and object models. Commun. ACM 42(1), 49–56 (1999)
14. Stahl, T., Volter, M.: Model-Driven Software Development. Wiley, Chichester (2006)
15. Williams, C., Kaplan, M., Klinger, T., Paradkar, A.M.: Toward engineered, useful use cases. Journal of Object Technology 4(6), 45–57 (2005)