Modeling and verification techniques for incremental development of UML architectures

Thanh-Liem Phan

École des mines d'Alès, LGI2P
Parc Scientifique Georges Besse, 30035 Nîmes cedex 1, France
phanliem@soict.hut.edu.vn
http://www.mines-ales.fr

Abstract. Our goal is to assist system architectural model development by providing techniques and tools to early detect specification and design errors. The models being developed deal with behavioral aspects of reactive systems: we wonder whether liveness properties are preserved during the model developments. For that purpose, we propose incremental modeling operations supported by formal relations to compare increments. The chosen relations are based on the conformance relation formally defined on labeled transition systems.

Keywords: incremental construction, architectures, UML, verification

1 Problems and objectives

We are interested in the development of architectural models of critical reactive systems expressed as assemblies of UML components. In order to specify and design architectures and behaviors, we advocate the use of an incremental development approach and model evaluations during the development process.

Refinement techniques offer several benefits. In particular, they integrate formal developments into the design and implementation process instead of conducting them in parallel, which reduces costs and improves consistency. However, classical refinement techniques do not allow us to follow an Agile method in which requirements are not only refined but also extended in order to offer new services or new functions. Consequently, they must start from initial abstract specifications which cover the whole system. This has two drawbacks: initial models are tricky to design; concrete implementations arrive lately which does not provide rapid feedback to clients and increases the stress of development teams.

The incremental development approach aims at responding to the limitations above by integrating both refinement and extension techniques: initial specifications can be partial; concrete implementation can be provided quickly which means feedbacks to clients are considered early. It appears that there isn't any technique that formally support the incremental construction of UML models, especially for reactive systems. The reactive properties are formalized by liveness properties (if a system is solicited on specific events, will it respond by the expected actions?). The verifications that we propose are conducted by binary

relations between models to know if the second one is actually a refinement, an extension, or an increment of the first one. All these relations guarantee the preservation of liveness properties. We propose a framework, IDCM [7] (Incremental Development of Conforming Models), in order to verify the above relations between UML models (state machines or composite structures).

2 Proposal: incremental construction of UML architectures

The proposed incremental construction approach for developing architectures in UML consists in *informal aspects* and *formal aspects*. Informal aspects cover the step by step development of architectures using UML notations. Formal aspects mean that software architects can compare at each development step the obtained model with respect to its previous step. Formal semantics that we have chosen is LTS (Labeled Transition Systems). In our incremental construction approach, informal aspect is realized by the toolkit TOPCASED [4], while formal aspect is implemented by our tool IDCM [7]. The overview of this approach is shown in Fig 1.

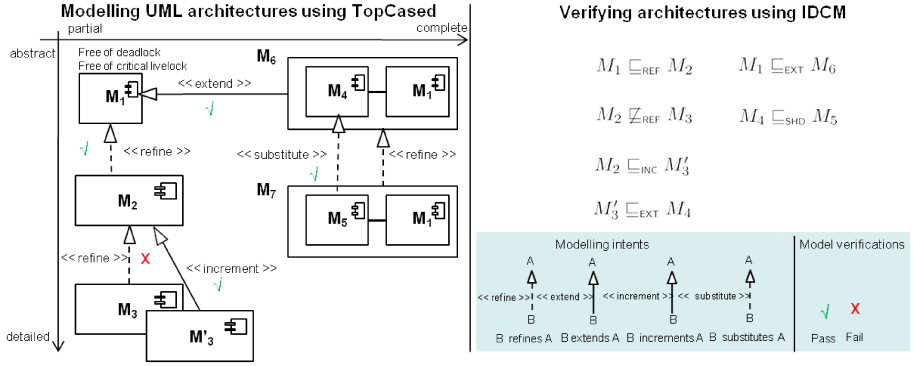

Fig. 1. Incremental construction approach of UML architectures.

The incremental construction of an architecture starts from a first model which is verified as being deadlock-free and without critical live-locks. The model is developed by adding, substituting, splitting components or reconfiguring architectures. In order to realize this approach, we consider two problems: **i) techniques for the construction of UML architectures**, and **ii) semantics and evaluations of UML architectures**. Evaluations are performed through the preorders shown in Fig. 1. We present the conformance relations proposed in [2] and introduce the should-testing preorder, which support compositional reasoning in software architecture, as follows.

Definition 1. *Let P and Q be two components and $\mathcal{A}$ the set of all operations of P and Q:*

- $Tr(P)$ *is the set of traces of* P. *A trace is a partial observable execution.*
- Q conf P (Q *conforms to* P) *if for all* $\sigma \in Tr(P)$, *for all* $A \subseteq \mathcal{A}$, P must accept A after $\sigma \Rightarrow Q$ must accept A after σ.
- $P \sqsubseteq_{\mathsf{EXT}} Q$ (Q *extends* P) *if* $Tr(P) \subseteq Tr(Q)$ *and* Q conf P.
- $P \sqsubseteq_{\mathsf{RED}} Q$ (Q *reduces* P) *if* $Tr(Q) \subseteq Tr(P)$ *and* Q conf P.
- $P \sqsubseteq_{\mathsf{INC}} Q$ (Q *increments* P) *if for any* I *such that* I conf Q, I conf P.
- $P \sqsubseteq_{\mathsf{REF}} Q$ (Q *refines* P) *if* $P \sqsubseteq_{\mathsf{RED}} Q$ *and* $P \sqsubseteq_{\mathsf{INC}} Q$.
- $P \sqsubseteq_{\mathsf{SHD}} Q$ (Q *can substitute* P) *if for all components* t *describing a test,* P should accept $t \Rightarrow Q$ should accept t.

P must accept A after σ refers to the set of sets of actions that P must accept after a trace σ [2]. If Q conf P, then Q is more deterministic than P and all liveness properties of P are satisfied by Q. P should accept t implies every reachable state of $P||t$ (P and t are executed concurrently) is required to be on a path to success [9]. If $P \sqsubseteq_{\mathsf{SHD}} Q$, then $P \sqsubseteq_{\mathsf{RED}} Q$ and moreover P can be replaced by Q in any hiding and parallel composition context.

In the case of substitution of components, none of the extension or refinement relations between the new component and the substituted component ensures the conformance of architectures. Then, we select the should-testing relation [1, 9], a congruence relation stronger than the refinement relation.

We give a simple example for illustrating the benefits of $\sqsubseteq_{\mathsf{SHD}}$ compare with $\sqsubseteq_{\mathsf{EXT}}$. Consider two components C_1 and C_2 in Fig. 2. We have $C_1 \sqsubseteq_{\mathsf{EXT}} C_2$ but when we substitute C_1 by C_2 in an architecture, we have the unwanted result $A_1 \not\sqsubseteq_{\mathsf{EXT}} A_2$. By using the relation $\sqsubseteq_{\mathsf{SHD}}$, we detect that $C_1 \not\sqsubseteq_{\mathsf{SHD}} C_2$.

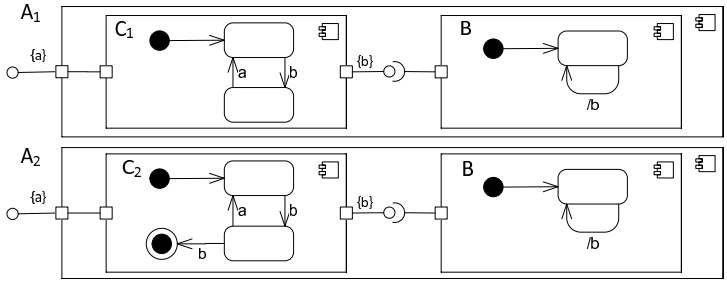

Fig. 2. $C_1 \sqsubseteq_{\mathsf{EXT}} C_2$ but $A_1 \not\sqsubseteq_{\mathsf{EXT}} A_2$.

2.1 Techniques for construction of UML architectures

In our approach, architectures can be developed following two axes: the *vertical axis* represents the level of abstraction, whereas the *horizontal axis* represents the level of requirements coverage. $\sqsubseteq_{\mathsf{REF}}$ is used for vertical techniques, $\sqsubseteq_{\mathsf{EXT}}$ is chosen for horizontal techniques, while $\sqsubseteq_{\mathsf{INC}}$ is a combination of both.

Vertical techniques. The construction of architectures along the vertical axis does not change the functionality of the system in terms of mandatory offered or required services, but the level of abstraction of the description. We present here some of these techniques:

- Refinement techniques (along the downward direction):
  - *Refinement solving* to find a component satisfying constraints to develop an architecture with reduced services. Let C a component to be refined using an existing component C_1. *Refinement solving* consists in developing or finding a component X and an architectural configuration f such that $A := f(C_1, X)$ satisfies $C \sqsubseteq_{\mathsf{REF}} A$.
  - *Refine substitution* to replace components in an architecture by refined ones. For an architecture $A_1 := f(C_1, ..., C_i, ..., C_n)$, *refine substitution* consists in finding a component C_i' such that $A_2 := f(C_1, ..., C_i', ..., C_n)$ satisfies $A_1 \sqsubseteq_{\mathsf{REF}} A_2$.
- Abstraction techniques (along the upward direction):
  - *Abstraction solving* to find components satisfying constraints to obtain an architecture more abstract. Let C a component to be abstracted using an existing component C_1. *Abstraction solving* consists in developing or finding a component X and an architectural configuration f such that $A := f(C_1, X)$ satisfies $A \sqsubseteq_{\mathsf{REF}} C$.
  - *Abstract substitution* to replace components in an architecture by other abstract ones.

Horizontal techniques. The construction of architectures along the horizontal axis does not change the level of abstraction of the model but involves the modification of behavioral specifications that can be realized through the offered and required services and the interactions with the environment. Main horizontal techniques are:

- Extension techniques, whose main modeling operations are:
  - *Extension solving* to find components which satisfy constraints to develop an architecture offering more services. For an architecture $A_1 := f(C_1, \ldots, C_n)$, *extension solving* consists in finding $g, C_1', \ldots, C_m'$ such that $A_2 := g(C_1', \ldots, C_m')$ satisfies $A_1 \sqsubseteq_{\mathsf{EXT}} A_2$.
  - *Extension substitution* to replace a component by another one which satisfies constraints to obtain an architecture offering more services. For $A_1 := f(C_1, \ldots, C_i, \ldots, C_n)$, *extensive substitution* consists in finding a component C_i' such that $A_2 := f(C_1, \ldots, C_i', \ldots, C_n)$ satisfies $A_1 \sqsubseteq_{\mathsf{EXT}} A_2$.
- Restriction techniques, whose modeling operations are *restriction solving* and *restriction substitution*. These techniques are the opposite of extension techniques and result in removing some behaviors or reducing interfaces.

2.2 Semantics and evaluation of UML models

To analyze components and architectures, their formal semantics need to be defined. We have automated two transformations of models developed using the Topcased framework [4]: the transformation of UML state machines into LTS and the transformation of UML architectures in intermediate specifications defined by synchronization vectors from which LTS are automatically generated using the CADP toolbox [5].

In order to evaluate UML models, designers choose the appropriate relation (among extension, refinement, increment, and substitution) according to the modeling relationship expressed in the component diagram. In case of failures, the verdict is expressed by refusal sets given after a sequence of interactions. Then, designers can correct their models and continue the incremental development. Experiments about modeling, transformation and conformance analyses have been conducted on several case studies.

3 Conclusion

We have developed a JAVA tool allowing LTS transformation and verification of UML models. The value of the incremental development approach is to offer a compromise between pragmatic approaches such as Agile development, and formal approaches such as method B.

The contribution of this work is twofold: i) providing evaluation means associated to specialization and realization UML relationships; and ii) supporting a substitution relation between UML components.

References

1. Ed Brinksma, Arend Rensink, and Walter Vogler. Fair Testing. In Scott Smolka, editor, *CONCUR '95: Concurrency Theory*, volume 962, pages 313–327. Springer-Verlag, August 1995.
2. Ed Brinksma and Giuseppe Scollo. Formal Notions of Implementation and Conformance in LOTOS. Technical report, Twente University of technology, Enschede, December 1986.
3. Anne-Lise Courbis, Thomas Lambolais, Hong-Viet Luong, Thanh-Liem Phan, Christelle Urtado, Sylvain Vauttier, et al. A formal support for incremental behavior specification in agile development. *Proceedings of SEKE 2012*, 2012.
4. Patrick Farail, Pierre Gaufillet, Agusti Canals, Christophe Le Camus, David Sciamma, Pierre Michel, Xavier Crégut, and Marc Pantel. The TOPCASED project: a toolkit in open source for critical aeronautic systems design. *Embedded Real Time Software (ERTS)*, 2006.
5. Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2010: a toolbox for the construction and analysis of distributed processes. In *LNCS*, volume 6605 of *LNCS*, pages 372–387. Springer-Verlag, March 2011.
6. Guy Leduc. A framework based on implementation relations for implementing LOTOS specifications. *Computer Networks and ISDN Systems*, 25(1):23–41, August 1992.
7. Hong-Viet Luong, Anne-Lise Courbis, Thomas Lambolais, and Thanh-Liem Phan. IDCM:un outil d'analyse de composants et d'architectures dédié à la construction incrémentale. In *11èmes Journées Francophones sur les Approches Formelles dans l'Assistance au Développement de Logiciels*, pages 50–53, January 2012.
8. Hong-Viet Luong, Thomas Lambolais, and Anne-Lise Courbis. Implementation of the Conformance Relation for Incremental Development of Behavioural Models. In Krzysztof Czarnecki, editor, *MoDELS 2008*, volume 5301/2009 of *LNCS*, pages 356–370. Springer-Verlag, 2008.
9. A. Rensink and W. Vogler. Fair testing. *Information and Computation*, 205(2):125–198, 2007.