Incremental construction of architectural specification supported by behavioral verification

Thanh-Liem Phan[1], Anne-Lise Courbis[1], Thomas Lambolais[1], Thérèse Libourel[2]

[1] Laboratory LGI2P, Ecole des Mines d'Alès
Site EERIE, Parc Scientifique Georges Besses, 30035 Nimes Cedex 1 - France

[2] UMR ESPACE-DEV, Université de Montpéllier 2
161 rue Ada, 34095 Montpellier Cedex 5 - France

Abstract. Our goal is to study the incremental construction of architectural specifications. Techniques and tools will be provided to detect errors in the specification and design phases. This work distinguishes two kinds of component: primitive components and composite components. Components are described using UML, and their formal semantics are given by transforming a subset of UML into LTS. The verification techniques are based on existing comparison relations to ensure that each step preserves dynamic properties of previous steps. Verifying architectures is accomplished by checking the freedom of dead-locks and the substitutability of components.

Keywords: UML, LTS, state machine, activity, composite structure, incremental development, software architecture, dead-locks, substitutability.

1 Introduction

The construction of critical reactive software architectures, in which errors could have serious consequences on human life, environments or significant assets, is challenging. When constructing such architectures, one focus on behavioral analysis in order to detect communication problems such as dead-locks between components. Hence, two aspects are considered: **construction processes of architectures** and **evaluation techniques**.

First, to support the construction of architectures, we believe that using an incremental approach [3][10] is suitable. The incremental construction operations of architecture that we consider are the following: i) **addition operation** (adding a component or a connection into the architecture); ii) **removal operation** (removing a component or a connection from the architecture); iii) **substitution operation** (substituting a component by a new one); iv) **split operation** (split a component into subcomponents); v) **merge operation** (several components are merged into a component). At this stage, the addition and substitution operations are focused.

Second, to support the evaluation techniques, we have to deal with two problems:

i. Define the semantics of architectures by transforming UML architectures into formal languages. We have chosen LTSs (Labeled Transition Systems) for the semantics of UML architectures and components.

ii. Compare models in order to verify that a model preserves the necessary properties of the previous version by using pre-orders and equivalences. Conformance relations (conf, red, ext, conf-restr, cred, cext) [2], [8], [11], testing pre-orders ($\sqsubseteq_{may}, \sqsubseteq_{must}, \sqsubseteq_{te}$), and bi-simulations (=, ≈) have been considered and implemented in previous works. However, conformance relations and testing pre-orders do not preserve the substitution property in hiding contexts, while bi-simulations relations are claimed to be too strong. In complex system design, hiding and parallel composition are the most important contexts and have to be considered carefully. So we need to find appropriate relations to be used correctly in hiding and parallel composition contexts.

2 Methods for architecture analysis

2.1 Architecture modeling

Two kinds of components are distinguished: primitive components, and composite components. A primitive component may specify its behavior by itself, while a composite component contains the internal architecture so that its behavior is deduced from its sub-components' behaviors. In this work, architectures mean composite components.

Our work focuses on defining *reusable pieces* of models, which means that *pieces* are independent and well-encapsulated. So, we use the notion of port, which is means to ensure the encapsulation property. Ports may be behavioral or non-behavioral. Ports of primitive components are all behavioral ports. Requests received are directly forwarded from a provided port to the classifier behavior of the owning component. Requests received from ports are indicated by Triggers of State Machines. In order to represent demands, which are sent to required ports from the classifier behavior, we use Invocation Actions of Activities. From our point of view, the combination of State Machines and Activity is necessary for the encapsulation of primitive components.

Architectures are modeled by assemblies of components using UML Composite Structures. Ports of architectures are non-behavioral ports, as they act like a routing device to forward the messages to or from the sub-components. At this moment, we restrict to binary connections between ports.

2.2 Semantics of architectures

The semantics of primitive components is determined by transforming a subset of State Machines and Activity into LTSs [10][11].

We define the semantics of architecture by transforming UML Composite Structures into EXP.OPEN [7] specification by a set of rules. Then LTSs are generated by using facilities of the CADP toolkit [5]. In case the system contains many components, the obtained LTS can be very complex. We have proposed the methods to compare the LTSs of architectures based on their minimizations [10].

2.3 Verification of architectures

Once the semantics of architectures is determined, the following verification activities can be done: a) verifying the absent of dead-locks in architectures; b) verifying the conformance between architectures using comparison relations.

Dead-lock analysis.

We have proposed a compatibility relation [4][6][12], which can be used to guarantee the absent of dead-locks in architectures, between two LTSs. Because it is always possible to consider the context of a component as a set of components, which can be modeled by a unique LTS, the compatibility between a component and its context can be verified. However, this relation is not strong enough to guarantee the conformance between two architectures.

In case a new component C is added into the system, the compatibility between C and its environment E (the components that C is connected to) needs to be verified. A possible solution is to benefit the advantages of the substitution relations (to be discussed in the next part) by replacing E by the new composite EC created from E and C. As a result, the verification of the substitutability between E and EC guarantees the absent of dead-locks and the conformance between the two versions of architecture.

Concerning b), to compare two architectures, there are two ways [10]:

i. Global analysis: the behaviors of the whole architecture are computed, and then analyzed using the conformance relations we have implemented. This method can be used to evaluate all incremental operations. But a problem appears when the system becomes complex so that having the LTS of the whole architecture is space expensive. The second way could solve this limitation.
ii. Differential analysis: only the behaviors of the modified parts between two architectures are considered. For example: a component is substituted by a new one (or a group of components); or a group of components is substituted by a component. This leads to the problem of component substitutability.

Component substitutability.

The relations which satisfy the substitutability properties in any context are congruence relations. Congruence relations defined over the conformance relation are **cext** and **cred** [8]. However, these relations fail to be congruent in hiding contexts creating divergences [8][9], (i.e. an infinite sequence of internal actions) which means **cred** and **cext** are not appropriate in the context of components assembly.

We are interested in the fairness assumption, in which divergences are not always considered as catastrophic (as in [9]). Fairness assumptions mean that the system is not allowed to continuously favor some choices at the expense of others. Fairness is important in reactive systems. We have found that the should-testing pre-order [1], which is congruent in parallel composition and hiding context, is an answer to a long stated problem: the greatest congruence stronger than **conf**. **It exactly corresponds to**

what we are looking for and in addition, its decidability is represented in [13]. We have studied and implemented this relation, which has the complexity of $O(nm2^{3n+5m})$.

3 Conclusion

We have considered the usage of UML State Machines and Activity for describing primitive components, and UML Composite Structures for describing architectures. Then a set of rules has been proposed to transform UML architectures and components into formal semantics. We have added to our tool, IDCM [10] (Incremental Development of Conforming Model): i) the transformation of component's behaviors (described by State Machine and Activity) into LTS; ii) the transformation of UML Composite Structures into EXP.OPEN. Finally, the dead-lock detection and substitutability problems have been considered. The should-testing pre-order [13], which is suitable for the context of component substitutability, has been implemented.

For future works, we would like to: i) study problems of asynchronous communication between components, which often are used in web services applications; ii) formalize a framework for incremental construction of architectures;

References

1. Brinksma, E., Rensink, A., Vogler, W.: Fair Testing. International Conference on Concurrency theory. 962, 313-327 (1995).
2. Brinksma, E., Scollo, G.: Formal notions of implementation and conformance in LOTOS. Twente University of Technology (1986).
3. Courbis, A.-L., Lambolais, T., et al.: A formal support for incremental behavior specification in agile development. SEKE. (2012).
4. Courbis, A.-L., Lambolais, T., Luong, H.-V., Phan, T.-L.: Analyse de l'interopérabilité et de la conformité d'architectures logicielles. CAL. (2011).
5. Garavel, H., Mateescu, R., Lang, F., Serwe, W.: CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. CAV. p. 158-163 (2007).
6. Lambolais, T., Courbis, A.L., Luong, H.V., Phan, T.L.: Interoperability Analysis of Systems. IFAC. (2011).
7. Lang, F.: Exp.Open 2.0: A Flexible Tool Integrating Partial Order, Compositional, and On-the-fly Verification Methods. (2005).
8. Leduc, G.: A framework based on implementation relations for implementing LOTOS specifications. Computer Networks and ISDN Systems. 25, 1, 23-41 (1992).
9. Leduc, G.: Failure-based Congruences, Unfair Divergences and New Testing Theory. PSTV XIV. (1994).
10. Luong, H.-V.: Construction Incrémentale de Spécifications de Systèmes Critiques intégrant des Procédures de Vérification. Thesis. Université Paul Sabatier-Toulouse 3 (2010).
11. Luong, H.-V., Lambolais, T., Courbis, A.-L.: Implementation of the Conformance Relation for Incremental Development of Behavioural Models. MODELS. p. 356-370 (2008).
12. Phan, T.-L., Lambolais, T., Courbis, A.-L.: Aide au développement incrémental et à la vérification d'architectures logicielles. Lambda Mu 18. (2012).
13. Rensink, A., Vogler, W.: Fair testing. Information and Computation. 205, 125–198 (2007).