Safe Incremental Design of UML Architectures

Anne-Lise Courbis[1], Thomas Lambolais[1], and Thanh-Hung Nguyen[2]

[1]LGI2P, IMT Mines Alès, France. firstName.lastName@mines-ales.fr
[2]Hanoi University of Science and Technology, Hanoi, Vietnam. hungnt@soict.hust.edu.vn

Abstract

IDF is an Incremental Development Framework which supports the development and the verification of UML models for reactive systems. IDF offers refinement and extension techniques allowing liveness properties to be preserved during the model developments. Here, we improve the framework in order to analyze models from a safety point of view. For this purpose, we associate IDF with the experienced tools of safety analysis based on the BIP language by translating UML models into BIP. We demonstrate on a basic example the complementarity of liveness and safety analyses.

Keywords*: UML composite components, BIP architectures, conformance analysis, safety analysis, refinement, incremental development.*

1. Introduction

Designing UML models of software intensive reactive systems is recognized to be a tricky and crucial task. Reactivity means that such systems must continuously react to their environment, at a speed defined by this environment. It implies *liveness properties*, stating that the system will eventually react as it must. These systems are also dependable, so that reliability, availability and robustness are of primary importance. This implies *safety properties*, stating that undesired behaviors of the system will never happen.

There is a lack of support for UML designers in the processes of both setting up models and evaluating them. The novelty of our proposed approach is to consider at early steps of the design both liveness and safety properties. In a previous work [8, 15, 14], we have presented our Incremental Development Framework (IDF) and its associated tool IDCM (Incremental Development of Conformance Models). IDF supports the development and the evaluation of UML models for reactive systems. It deals with UML composite components whose parts are composite or primitive components. Primitive component behaviors are described by UML state machines. IDF allows models to be developed step by step. At every step of the design, the model is verified as being consistent with the model obtained at the previous step. A step is a model evolution which can be of four kinds: *extension*, *refinement*, *increment* or *substitution*. The verification of these four kinds of model evolution is based on a conformance relation [16], which ensures that liveness properties of the former modeling step are preserved. However, this work has its own shortcomings: explicit verification of safety properties is not addressed. This article aims at enhancing IDF in order to be able to model and check explicit safety properties. This way, all temporal properties are considered, since the safety/liveness spectrum covers all linear temporal logic properties.

We present in section 2 an example pointing out an incremental development of a model whose liveness analysis is demonstrated using IDCM, but suffering from a lack of safety analysis. Section 3 presents the BIP intermediate format and the UML to BIP transformation in order to use D-Finder tool to analyze safety properties. The IDCM tool associated with IDF, as well as UML and BIP models presented in this article, may be downloaded on the website [7]. Section 4 presents related works. We conclude in section 5 and present our future directions.

2. Motivating example

Let us consider MUTEX, a mutual exclusion system that performs two task execution orders in parallel: it has two ports (Fig. 1), each of them allowing the reception of a task execution (start operation of IUserIn interface) and the transmission of an acknowledgement at the end (finish operation of IUserOut interface). The high level specification of MUTEX describes the system from an external point of

DOI: 10.18293/SEKE2017-033

view. The resource is not represented yet. It is modeled from a behavioral point of view by an atomic UML component (named SpecMUTEX) whose behavior is specified by a state machine with a concurrent state modeling two parallel task processes (Fig. 1).

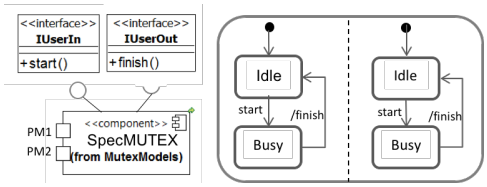
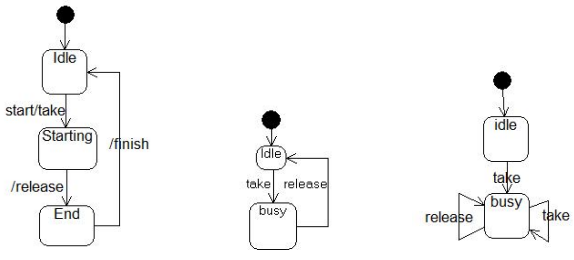


Figure 1: SpecMUTEX component

MUTEX1 model represents the first internal view of the system, i.e. two users sharing a resource. The resource provided interface has two operations: take and release. The designer thus specifies SpecUser and SpecResource components and their associated behaviors (Fig. 2a and b) in order to match SpecMUTEX specification. MUTEX1 is a composite component (Fig. 3a) which assembles two SpecUser components (U1 and U2) and one SpecResource component (R1) .

MUTEX2 model is a possible implementation model of MUTEX1 architecture. MUTEX2 has the same architecture than MUTEX1 (Fig. 3a) except that U1 and U2 are of type User and R1 is of type Resource (Fig. 2c). User details are not required to understand the example.

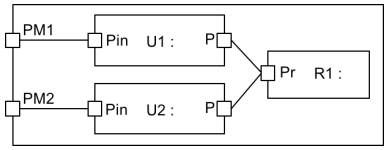
Both User and Resource are checked by IDCM as conforming their specification SpecUser and SpecResource (Fig. 3b). The conformance relation between an implementation model and a specification model guarantees that actions that are mandatory after any trace of the specification must also be accepted by the implementation after the same trace. This relation is implemented [16, 8] with its variant, refines and extends. It requires models to be transformed into LTS (Labelled Transition System). This is automatically achieved using IDCM [15, 14] which transforms the architecture into the EXP.OPEN formalism.



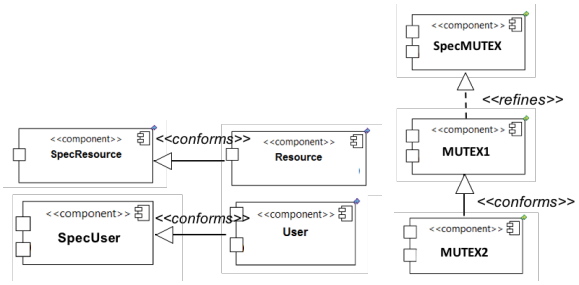
(a) SpecUser (b) SpecResource (c) Resource

Figure 2: MUTEX component state machines

Then, CADP [11] transforms EXP.OPEN into LTS. In the following, we give an interpretation of these relations on the MUTEX models (Fig. 3c). MUTEX2 component conforms to MUTEX1: it is a suitable implementation of the specification. MUTEX1 is a refinement of SpecMUTEX: it has no extra traces and must realize all mandatory behaviors of the specification.



(a) MUTEX1 and MUTEX2 architectures



(b) Implementation relations (c) Liveness relations

Figure 3: Incremental development of the MUTEX system

This example points out that MUTEX2 is a "good" realization of the initial specification from a liveness point of view, as defined by the ISO standard [12]. However, it does not verify the safety property stating that the resource has to be exclusive. This property states that U1 and U2 users must not at the same time be in the Starting state, which is the output state of the start/take transition, while R1 is in the Busy state .

It is necessary to associate another tool with IDCM to verify such a property.

3. Safety analysis of architectures

Our goal is to explicitly specify safety properties in order to complement incremental techniques. We are looking for a method which takes into account the incremental aspect of modeling in order to be integrated into IDF. The BIP (Behavior, Interaction and Priority) modeling and verification framework [2] is based on several principles which match with IDF: development of correctness-by-construction model, incrementality, compositionality and composability. It includes many tools for safety property analysis and model transformation from several languages such as AADL and Lustre. To the best of our knowledge, there is no available transformation tool from UML to BIP taking into account both primitive and composite compo-

nent descriptions. We have thus developed a module of IDCM that may be uploaded on the IDCM website [7]. We present an overview of the transformation from UML to BIP and the benefits for safety analysis of architectures.

3.1. From UML to BIP

BIP [1] is a powerful language for modeling heterogeneous real-time systems. Main classes of BIP meta-model corresponding to UML concepts we deal with are: Compound Type, Atomic Type, State, Transition, Port and Connector. BIP describes compound components by a set of interactions between atomic components whose behaviors are represented by LTS. There is thus a direct mapping between BIP and UML atomic components since we gave in [14] a LTS semantics to UML models. Listing 1 gives the corresponding BIP model automatically generated by the IDCM module UMLtoBIP.

```
package Resource
atomic type Resource
  export port Port PR_TAKE
  export port Port PR_RELEASE
  port Port i
  place Pseudostate1, idle, busy
  initial to Pseudostate1
  on i from Pseudostate1 to idle
  on PR_TAKE from idle to busy
  on PR_TAKE from busy to busy
  on PR_RELEASE from busy to busy
end
end
```

Listing 1: Resource BIP model

There is a direct mapping between UML composite component and BIP compound components: a UML composite component consists of a set of *Parts* which match BIP *Components*. A UML assembly *Connector* matches a set of *BIP connectors*. Indeed, a BIP connector is relative to the synchronization of a single operation shared by two interconnected ports, while a UML connector is relative to the synchronization of the set of operations belonging to interfaces associated with the interconnected ports. The *Port* of a *Part* belonging to a delegate connector will be exported and renamed by the name of the port of the compound component. To illustrate this transformation, we give in Listing 2 the BIP code of the MUTEX2 architecture presented in section 2. This code is automatically generated by the transformation IDCM module UMLtoBIP.

3.2. D-Finder: a toolbox for safety analysis

D-FINDER provides methods and tools to compute invariants of BIP models. Such invariants are interesting since they preserve safety properties. There are two kinds of invariants: component invariants which are over approximations of reachable states, and interaction invariants which

```
model Mutex
include User.bip
include Resource.bip
connector type RDV(Port p1, Port p2)
        define [p1 p2] end
compound type MutexType
  component Resource R1
  component User U2 component User U2
  connector RDV C1_release(U1.P_RELEASE, R1.PR_RELEASE)
  connector RDV C1_take(U1.P_TAKE, R1.PR_TAKE)
  connector RDV C2_release(U2.P_RELEASE, R1.PR_RELEASE)
  connector RDV C2_take(U2.P_TAKE, R1.PR_TAKE)
  export port Port PM1_FINISH is U1.PIN_FINISH
  export port Port PM1_START is U1.PIN_START
  export port Port PM2_FINISH is U2.PIN_FINISH
  export port Port PM2_START is U2.PIN_START
end
component MutexType Mutex
end
```

Listing 2: Mutex BIP model

define global boolean constraints dealing with the synchronization of components. D-FINDER is based on an abstraction technique allowing the state space to be reduced. Its strength is to perform incremental constructions of models and incremental computations of invariants [2] allowing large-scale systems to be checked. D-Finder uses the BDD library for the symbolic computation of interaction invariants, and then the SAT-solver tool Yices [9] for checking satisfiability. Verifying a safety property consists in demonstrating using Yices that the negation of the property is unsatisfiable in a context defined by the set of invariant expressions generated by D-Finder. The invariants are expressed by Boolean Behavioral Constraints [17].

3.3. Illustration of a safety property for MUTEX

Let us consider MUTEX1 and MUTEX2 architectures presented in section 2. We aim at checking the mutual exclusion property. Equation (1) expresses the non expected property: two users U1 and U2 can be both in Starting state while the shared resource R1 is in busy state.

$$(and\ \text{R1\_busy-}\ (and\ \text{U1\_Starting-}\ \text{U2\_Starting-})) \quad (1)$$

For MUTEX1, the property is unsatisfied: it means that the resource mutual exclusion has been properly implemented in this architecture. That is not the case for the MUTEX2 architecture. Indeed, the Resource state machine (Fig.2c) points out that it may be used concurrently by two users. On this example, the error is obvious, but it is not the case for large systems where components may be designed by third parties according to a high specification level.

4. Discussion and Related work

To the best of our knowledge, no framework support the *incremental* development of UML architecture mod-

els by analyzing both *liveness and safety* behavioral aspects. In particular, no framework is able to consider abstract and non-deterministic UML models and few frameworks are able to consider UML models partially covering the requirements. Hence, even if some work addresses refinement of models, they do not focus on the reduction of non-determinism, and most of them cannot analyze models the specification of which is extended, despite it is a key action for designing complex systems and managing model evolution. For example, [13] defines a UML profile to transform models into Wright for using the FDR model checker [10].[18] proposes to translate architectures into IF/IFx models [5] allowing LTS models to be generated and analyzed by the CADP model checker [11].Other approaches about AADL aims at transforming models into intermediate models such as FIACRE [6, 3] or BIP [6] to use appropriate model checkers such as TINA [4] or Yices [9].

Refer to [14] to have more arguments and a complete state of the art about formal verification of models.

These approaches are powerful from the safety point of view but they are not able to integrate liveness analysis for incremental development of models as it is done in IDCM.

5. Conclusion

In this article, we have pointed out the interest for incremental development of UML models and the complementarity between safety and liveness analyses. We have defined a transformation of UML models into the BIP formalism which is implemented into the tool IDCM we have developed. By this way, the liveness and safety analyses are automated. Safety properties are expressed by propositional calculus and requires designers to manipulate a specific syntax. Further steps consist in developing a support to help designers to express the safety properties in terms of UML concepts regardless the theorem prover syntax, and studying their automatic rewording when UML models are refined or extended.

Acknowledgement: This research was supported by IMT Mines Alès–France through its mobility funding program and the National Foundation for Science and Technology Development (NAFOSTED) under Grant 102.03-2013.39: Automated verification and error localization methods for component-based software.

References

[1] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In *SEFM*, pages 3–12, 2006.

[2] S. Bensalem, M. Bozga, A. Legay, T.-H. Nguyen, J. Sifakis, and R. Yan. Incremental component-based construction and verification using invariants. In *FMCAD*, pages 257–266, 2010.

[3] B. Berthomieu and J.-P. Bodeveix. Formal Verification of AADL models with Fiacre and Tina. In *ERTS*, 2010.

[4] B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA: Construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, (14):2741–2756, 2004.

[5] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF Toolset. In *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *LNCS*, pages 237–267. Springer Berlin Heidelberg, 2004.

[6] M. Y. Chkouri and M. Bozga. Prototyping of distributed embedded systems using AADL. *ACESMB*, pages 65–79, 2009.

[7] A.-L. Courbis, T. Lambolais, H.-V. Luong, and T.-L. Phan. IDCM. `http://idcm.wp.mines-telecom.fr`. Accessed: 2017-05-05.

[8] A.-L. Courbis, T. Lambolais, H.-V. Luong, T.-L. Phan, C. Urtado, and S. Vauttier. A formal support for incremental behavior specification in agile development. In *The 24th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 694–699, 2012.

[9] B. Dutertre. Yices 2.2. In A. Biere and R. Bloem, editors, *CAV*, volume 8559 of *LNCS*, pages 737–744. Springer, July 2014.

[10] Formal-Systems and Oxford-University-Computing-Laboratory. Failures-Divergence Refinement (FDR2 User Manual). Technical Report October, Formal System (Europe) Ltd, 2010.

[11] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2011: a toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer*, 15(2):89–107, 2013.

[12] ISO/IEC9646. Information technology – open systems interconnection – conformance testing methodology and framework – part 1: General concepts, 1991.

[13] M. Kmimech, M. T. Bhiri, and P. Aniorte. Checking component assembly in ACME: an approach applied on UML 2.0 components model. In *ICSEA*, pages 494–499. IEEE, 2009.

[14] T. Lambolais, A.-L. Courbis, H.-V. Luong, and C. Percebois. IDF: A framework for the incremental development and conformance verification of UML active primitive components. *Journal of Systems and Software*, 113:275–295, 2016.

[15] T. Lambolais, A.-L. Courbis, H.-V. Luong, and T.-L. Phan. Designing and integrating complex systems: Be agile through liveness verification and abstraction. In *CSDM*, pages 69–81. Springer, 2015.

[16] H.-V. Luong, T. Lambolais, and A.-L. Courbis. Implementation of the Conformance Relation for Incremental Development of Behavioural Models. In *MoDELS*, volume 5301 of *LNCS*, pages 356–370. Springer Berlin, 2008.

[17] T.-H. Nguyen. *Constructive verification for component-based systems*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2010.

[18] I. Ober and I. Dragomir. Unambiguous UML composite structures: the OMEGA2 experience. In *SOFSEM*, pages 418–430. Springer, 2011.