EAI Endorsed Transactions
on Context-aware Systems and Applications
Research Article ICST.ORG

Context-aware approach for formal verification

Amel Benabbou[1,*], Safia Nait Bahloul[1] and Dhaussy Philippe[2]

[1]LITIO Laboratory, IDTW team, University of Oran 1 Ahmed Ben Bella, BP 1524, El-M'Naouer, 31000 Oran, Algeria
[2]Lab-STICC Laboratory, MOCS team, ENSTA-Bretagne, France

Abstract

The Context-aware approach has proven to be an effective technique for software model-checking verification. It focuses on the explicit modelling of environment as one or more contexts. In this area, specifying precise requirement is a challenged task for engineer since often environmental conditions lack of precision. A DSL, called CDL, has been proposed to facilitate the specification of requirement and context. However, such language is still low-level and error prone, difficult to grasp on complex models and assessment about its usability is still mitigated. In this paper, we propose a high level formalism of CDL to facilitate specifying contexts based on interaction overview diagrams that orchestrate activity diagrams automatically transformed from textual use cases. Our approach highlights the boundaries between the system and its environment. It is qualified as model-checking context-aware that aims to reduce the semantic gap between informal and formal requirements, hence the objective is to assist and encourage engineers to put sufficient details to accomplish effectively the specification process.

Keywords: Context, Context-awareness, Context-aware verification, Model-checking, Model transformation, Use cases, Interaction overview diagram.

Received on DD MM YYYY, accepted on DD MM YYYY, published on DD MM YYYY

doi: 10.4108/_____

1. Introduction

Describing the behaviour of systems is the principle on which verification by model-checking is based. In such technique, the System Under Study (SUS) is abstracted as a model presented, generally, in the form of concurrent automata (state machine) and on which we aim to verify the correctness of requirements expressed in a formal properties language. The whole of the model behaviours (i.e., states) are explored by the model-checker to evaluate whether the specified properties are true or not. During exploring model states, the number of reachable configurations is become too large to be contained in the memory. This is known by the state explosion problem [1].

To overcome this problem, many works proposed to improve the performance of model-checkers by considering compositional verification [2, 3, 4, 5]. In this area, context-aware verification has been introduced [6, 7] as a technique of state space decomposition that enables compositional verification of requirements. The idea is to allow to explicit separately the behaviour of entities (actors) that interact with the system and its environment. This technique reduces the set of possible spaces behaviours (and thus the state space) by considering an explicit model of the environment during its exploration. It consists to "close" the SUS with a well defined finite and acyclic environment. The reduction is based on the description of particular use cases on the environment, called contexts, with which the system interacts. The objective is to guide the model-checker to concentrate its efforts not on an exhaustive exploration of the whole model but on a relevant restriction of this latter. The formal specification of the environment enables at least three different decomposition axes: a) the environment can be decomposed in contexts; b) contexts enable the automatic partitioning of the state space into independent verification

*Corresponding author. Email: benabbou_amel@yahoo.fr

$Wait\ (0; C_2)$ $Wait\ (C_2)$
$Wait\ (C_1||C_2)$ $Wait\ (C_1) \cup Wait\ (C_2)$

A context is considered as a process communicating asynchronously with the system. Its input events are memorized in a buffer. The semantics of CDL is defined by the relation (C, B) $\xrightarrow{a}$ (C′, B′) to express that the context *C* with the buffer B "produces" *a* (which can be a sending or a receiving signal, or the nullσ signal if *C* does not evolve) and then becomes the new context *C′* with the new buffer B′. This relation is defined by the following 8 rules (In these rules, a represents an event which is different from $nul_{l\sigma}$):

// An MSC beginning with a sending event a! emits this event and continues with the remaining MSC.

$$\frac{}{(a!; M, B) \xrightarrow{a!} (M, B)} \quad [pref1]$$

// Expresses that if an MSC begins by a reception *a*? and faces an input buffer containing this event at the head of the buffer, the *MSC* consumes this event and continues with the remaining *MSC*.

$$\frac{}{(a!; M, a.B) \xrightarrow{a?} (M, B)} \quad [pref2]$$

// Establishes that a sequence of contexts $C_1; C_2$ behaves as C_1 until it has terminated.

$$\frac{C_1' \neq 0 \quad (C_1, B) \xrightarrow{a} (C_1', B')}{(C_1.C_2, B) \xrightarrow{a} (C_1'.C_2, B')} \quad [seq1]$$

// If the first context C_1 terminates (i.e., becomes 0), then the sequence becomes C_2.

$$\frac{(C_1, B) \xrightarrow{a} (0, B')}{(C_1.C_2, B) \xrightarrow{a} (C_2, B')} \quad [seq2]$$

// the semantics of the parallel operation is based on an asynchronous interleaving semantics

$$\frac{C_1' \neq 0 \quad (C_1, B) \xrightarrow{a} (C_1', B')}{\begin{array}{l}(C_1||C2, B) \xrightarrow{a} (C_1'||C_2, B') \\ (C_2||C1, B) \xrightarrow{a} (C_2||C1, B')\end{array}} \quad [par1]$$

$$\frac{(C_1, B) \xrightarrow{a} (0, B')}{\begin{array}{l}(C_1||C_2, B) \xrightarrow{a} (C_2, B') \\ (C_2||C1, B) \xrightarrow{a} (C_2, B')\end{array}} \quad [par2]$$

//The alternative context $C_1 + C_2$ behaves either as C_1 or as C_2.

$$\frac{(C_1, B) \xrightarrow{a} (C', B')}{(C_1 + C_2, B) \xrightarrow{a} (C_1', B')} \quad [alt]$$

$$(C_2 + C_1, B) \xrightarrow{a} (C_1', B')$$

// If an event *a* at the head of the input buffer is not expected, then this event is lost.

$$\frac{a \notin wait(C)}{(C, a.B) \xrightarrow{null\sigma} (C, B)} \quad [discard_C]$$

For more description of CDL language, see the published articles [7, 8, 9] available on http: //www.obpcdl.org

3. Problem statement and objectives

Use cases are a key element in our context-aware approach. Traditionally, used in capturing requirements, they are efficient to uncover, through scenarios, the behaviour of actors and to help focusing on their interactions with the system.

Within a CDL specification, the behaviour of each actor is considered as series of scenarios. These behaviours are composed in parallel to generate all the possible sequences of events. Thus, users are required to identify the behaviour of each actor to formalize it in the form of a CDL scenario. This is a manual process that requires: a) significant effort, to make the connection between the both modelling levels (use case and CDL), especially when the system is strongly coupled with its environment; b) good knowledge of the syntax and semantic of CDL. There is a semantic gap between the textual descriptions of use cases describing scenarios and CDL_S models that capture sent and received messages by each actor. Moreover, produce an exhaustive description of events seems to be a complicated task because CDL is based on simple scenarios, which are just partial set of interactions.

CDL has been evaluated through several aeronautic and military industrial case studies [13]. However, industrial feedback reports that although CDL has solved several state explosion cases, it is perceived as a low-level language, restrictive and difficult to grasp and apply on complex models. Then, we need to express environmental scenarios at a higher level of abstraction that maps better to requirement and specification engineers. The new UML interaction diagrams are suitable for high-level specifications. We use IODs known by their ability to show the control flow with a sequence of more general interactions [14]. IODs constitute a high-level structuring mechanism that we use to synthesize scenarios. In our approach, IODs are used to: i) capture the behaviour of the system, ii) describe the messages flow in the system and iii) describe the structural organization of CDL.

Our approach is qualified as model-checking context-aware. This aims to facilitate contexts elaboration and to build intermediate models between use cases and CDL, allowing the automatic generation of CDL models from the manipulated artefacts. The main objective of the whole approach is to assist and encourage engineers to put sufficient details to accomplish effectively the specification process.

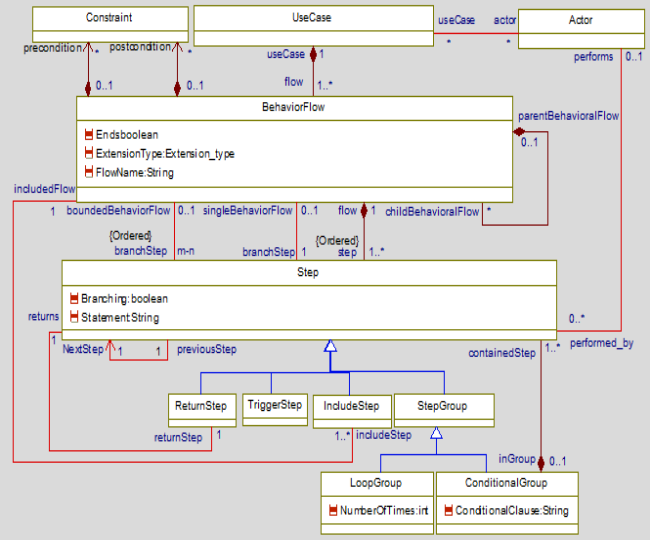


Figure 4. Use case Meta-model

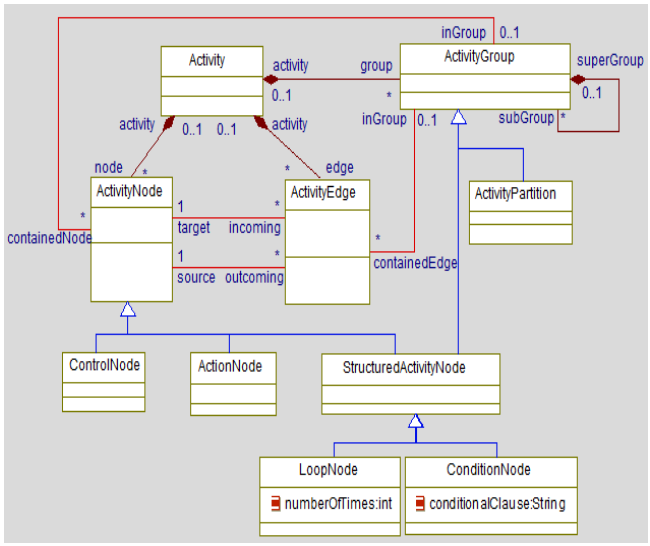


Figure 5. Activity Meta-model

A use case is associated to one or many scenarios, called *BehaviourFlow*, some of them are main scenarios. A *BehaviourFlow* is made of an ordered sequence of Steps: *SingleStep* or *StepGroup*. A *StepGroup* contains an ordered sequence of *Step*, including other *StepGroups* recursively. A *SingleStep* may be specialized in: *TriggerStep* (the condition triggering a *BehaviourFlow*), *IncludeStep* (the step contains another *BehaviourFlow*), *ReturnStep* (a return to another Step), *FinalStep* (the use case ends). A *StepGroup* is a *LoopGroup* or a *ConditionalGroup*. A *BehaviourFlow* can have extension(s) (alternatives that describe different steps than those in a success scenario) and it applies recursively.

A *child BehaviourFlow* refers to a parent *BehaviourFlow* and states the branching point where the extension condition (a *TriggerStep*) should be checked: a single branching point or a bounded interval; in the latter case, the condition can occur at any steps within the bounds and triggers the *child BehaviourFlow*.

The second meta-model that we use is that of activity diagram. In this meta-model, *Activity* is a generalization of *ActivityNodes* and *ActivityEdges* for linking between them. *ActivityNode* is either a simple *action*, a *ControlNode (decision, fusion, etc)* or some specialization of groups of *StructuredActivity* in *looped* and *conditional* forms. An

ActivitygGroup generalizes also the *partition* notion that gathers activities for each actor. Activity meta-model is given as follows:

5.2. A running model-checking example

We use a famous concurrency problem to illustrate a typical model-checking process, the context-aware approach and to introduce our proposal later. It's about Lamport's problem of two neighbours Alice and Bob that share a yard in an exclusive manner [15]. This problem is presented within the following algorithm:

```
Alice :
        while ( true )      {
        f l a g A l i c e = up ;
                while    ( flagBob == up)  skip ;
        catInYard ;
        f l a g A l i c e = down ;
        }
Bob :
        while ( true ) {
        flagBob = up ;
                while    ( f l a g A l i c e == up)   {
                flagBob = down ;
                        while ( f l a g A l i c e   == up) skip ;
                flagBob = up ; }
        dogInYard ;
         flagBob = down ; }
```

According to our context-aware verification approach, we need the followings artefacts: i) the system is translated into specification model to describe the behaviours of *Alice* and *Bob*, in the form of automata given in left side (A) in Figure 6. The expression evRain [*catIn-Yard* = true / *AliceCatGoesHome*] means that when the event *evRain* occurs and if the condition *catInYard* = true is satisfied then the action *AliceCatGoesHome* is performed. ii) Contexts are given through use cases "*Alice's cat comes home*" and "*Bob releases a dog*" given in middle (B) and right sides (C), respectively. iii) A property to be checked, formalized using CDL, for instance the mutual exclusion property may be represented with not (*catInYard* and *dogInYard*).

Apply rule AER3: - Generates 2 *ActivityEdges* for the *ActionNode Alice opens the door to her cat* and 1 *ActivityEdge* for the *ActionNode Alice asks the system to lower her flag*. These *ActivityEdges* link together the *DecisionControlNodes* after an *ActionNode* and the last *DecisionNode* either to a *FusionControlNode* (case of the former *ActionNode*) or to the *ActionNode* (latter case).
Apply rule AER4:- Generates 2 *ActivityEdges* sourcing from the 2 *ActionNodes* that are non-ending steps and targeting the first *ActivityNode* of each following *ActionNode* (a *DecisionNode* in both cases).

Figure 7 shows the generated activity diagram as follow:

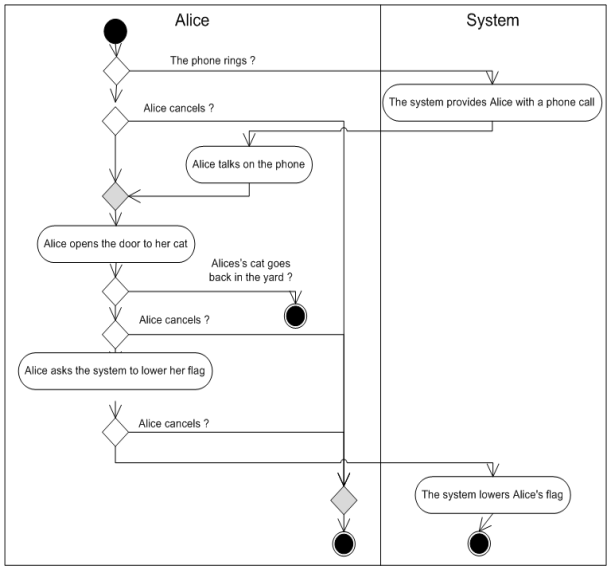


Figure 7. The generated activity diagram for "*Alice's Cat goes home*" use case

5.4. Specification of the system boundaries

Our aim now is to transform the resulting activity diagrams to our first type of IODs, focusing only on the actor's partition and its interactions with the system. To do this, we need to use boundaries to establish the interface (focusing on exchanged messages) between the system and its environment. Our IODs are established as follows: first, we recommend writing actions with simple sentences having a subject, a verb, and eventually an object. Actions without the system as a subject or an object (such as *Alice opens the door to her cat*) are out of the scope and will be discarded. Compound actions (such as *Alice releases her cat* and *warns the system*) have to be split in simple actions (such as *Alice releases her cat* - out of the scope - and *Alice warns the system*) - within the scope). When the simple sentence rule is applied, it is easy to process *ActionNodes* and recognize if the system is a subject or an object and eventually discard the *ActionNode* from the system scope. The same rule applies to *DecisionControlNodes*: if the condition includes any reference to the system, the *DecisionControlNode* will be kept, else discarded. Any incoming or outgoing *ActivityEdges* to a discarded *ActivityNode* (*Action* or *Decision*) will be discarded too, and the pending *ActivityEdge* reconnected to the following *ActivityNode* (that might be discarded later, forcing the *ActivityEdge* to be reconnected).

At the end, a set of nodes are discarded. Moreover, there are *ActivityEdges* crossing the boundary and because the system partition is not included, such *ActivityEdge* will be cut and replaced by a pair of related gates, one is the actor's model and another is the system model. Thus, an IOD is established for each actor (*Actor IOD*). See Figure 8:

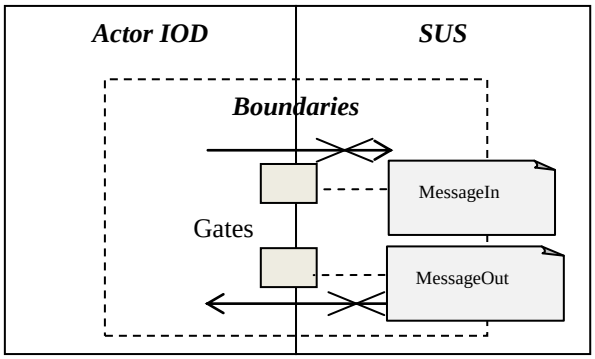
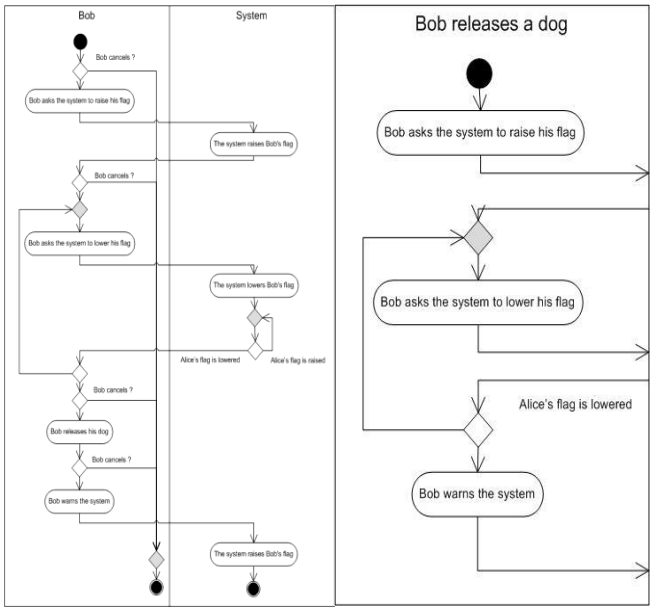


Figure 8. IOD and system boundaries

The activity diagram (left side (A)) and its IOD (right side (B)) generated from the use case "*Bob releases a dog*" are given in Figure 9:



(A) (B)

Figure 9. IOD corresponding to the use case "Bob releases a dog" and System Boundaries

transformation activities, the semantic gap between informal and formal requirements is reduced and engineers helped towards formal verification.

As [26], our approach is still manual. However, we aim to automate the transformation process for a validation purpose on an industrial case study to check the completeness and correctness of our transformation rules. A framework for automation like Ecore for meta-models implementation and Java for rules transformation are among proposals. However, we are looking for a solution that might be easily customized to the tools set used by the users: XML Metadata Interchange (XMI) enables interchange of meta-data between UML-based modelling tools but there might be slightly differences between the tools meta-models.

References

[1] Pelanek, R. (2009) " Fighting state space explosion: Review and evaluation". In Formal Methods for Industrial Critical Systems, volume 5596, pages 37{52. Springer Berlin Heidelberg, 2009.

[2] Alur, R., Brayton, R., Henzinger, T., Qadeer, S. and Rajamani, S. (1997) " Partial-order reduction in symbolic state space exploration". In Computer Aided Veri_cation, volume 1254, pages 340_351. Springer Verlag, LNCS, 1997.

[3] Valmari, A. (1991) "Stubborn sets for reduced state space generation". In Proceedings of the 10th International Conference on Applications and Theory of Petri Nets, pages 491_515, London, UK, 1991. Springer-Verlag.

[4] Park,. S and Kwon, G. (2006). "Avoidance of state explosion using dependency analysis in model-checking control flow model". In Proceedings of the 5th International Conference on Computational Science and Its Applications (ICCSA '06), volume 3984, pages 905_911. Springer-Verlag, LNCS, 2006

[5] Bosnacki, D. and Holzmann, G. (2005) " Improving spin's partial-order reduction for breadth-First search". In SPIN2005, volume 3639, pages 91_105, 2005.

[6] Dhaussy P, Boniol, F. and Roger, J. (2011a). "*Reducing State Explosion with Context Modeling for Model-Checking*. In 13th IEEE International High Assurance Systems Engineering Symposium (Hase'11), Boca Raton, USA, 2011. 5, 37, 38, 39, 40, 44.

[7] Dhaussy, P. Boniol, F. Roger, J. and Leroux, L. (2012a). Improving model-checking with context modeling. Advances in Software Engineering, ID 547157:13 pages, 2012.

[8] Dhaussy, P. and Roger, J. "*CDL (Context Description Language)* : Syntax and Semantics". Rapport technique, ENSTA- Bretagne, 2011. 37

[9] Dhaussy, P., Roger, J., Leroux, L. and Boniol, F. "Context Aware Model Exploration with OBP tool to Improve Model-Checking". ERTS'12, February 1-3, 2012.

[10] Chaelynne M. Wolak, (2001) "Gathering Requirements: The Use Case Approach". School of Computer and Information Sciences , Nova Southeastern University ,June 2001

[11] Whittle, J. (2006) "Specifying precise use cases with use case charts". In Proceedings of the 2005 International Conference on Satellite Events at the MoDELS, pages 290{301. Springer-Verlag, 2006.

[12] Dhaussy, P., Pillain, P., Creff, S., Raji ,A., Le Traon, Y. and Baudry, B.(2009) "Evaluating context descriptions and property definition patterns for software formal validation". In 12th IEEE/ACM conference on Model Driven Engineering Languages and Systems (Models'09), volume 5795, pages 438_452. Springer-Verlag, LNCS, 2009.

[13] OMG UML. "OMG unified modeling language™, infrastructure". Technical report, Object Management Group, (http://www.omg.org/spec/UML/)

[14] Berthomieu, J., Bodeveix, JP., Farail, P., Filali, M., Garavel, H., Gaufillet, P., Lang, F. and .Vernadat, F. (2008) " Fiacre: an intermediate language for model verification in the topcased environment". In *ERTS 2008*. 2008.

[15] Lamport, L. (1983) "invited address solved problems, unsolved problems and non-problems in concurrency". In Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, pages 1{11. ACM, 1984.

[16] Benabbou A. (2015) "Formalisation des interactions et des exigences pour la génération des modèles cdl "- partie 1 : Contextes. Technical Report 2015-03-01, ENSTA Bretagne.

[17] E. M. Clarke, D. E. Long, and K. L. Mcmillan, Compositional Model Checking, MIT Press, 1999.

[18] L. De Alfaro and T. A. Henzinger, "Interface automata," in Proceedings of the 8th Eiropean Engineering Conference and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE '01), pp. 109–120, ACM Press, September 2001.

[19] Cormac, F. and Shaz Q.(2003)" Thread-modular model checking". In SPIN'03, 2003.

[20] Oksana, T. and Matthew, D.(2003)" Automated environment generation for software model checking". In Proceedings of the 18th International Conference on Automated Software Engineering, pages 116–129, 2003.

[21] Pnueli, A. (1977) "The temporal logic of programs". In SFCS '77: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, pages 46_57, Washington, DC, USA, 1977. IEEE Computer Society.

[22] Clarke, E., Emerson, E. and Sistla, A. (1986) "Automatic verification of finite-state concurrent systems using temporal logic specifications". ACM Trans. Program. Lang. Syst., 8(2):244_263, 1986.

[23] Almendros, J. and Iribarne, L. (2005) "Describing use cases with activity charts". In *Metainformatics*, volume 3511, pages 141–159. Springer Berlin Heidel- berg, 2005.

[24] Gutierrez, C., Nebut, M., Escalona, M., Mejas, I. and Ramos, M. (2008) "Visualization of use cases through automatically generated activity diagrams". In Model Driven Engineering Languages and Systems, pages 83{96. Springer Berlin Heidelberg, 2008.

[25] Tao, Y., Lionel, C., Briand, and Yvan, L. (2010) "An automated approach to trans-form use cases into activity diagrams". In Modeling Foundations and Applications, pages 337{353. Springer, 2010.

[26] Mustafiz, S., Kienzle, J., and Vangheluwe, H.(2009) "Model transformation of dependability-focused requirements models". In ICSE Workshop on Modeling in Software Engineering, MISE '09, pages 50{55. 2009.

[27] Sharma, R., Gulia, S., Biswas, K. (1977)" Automated Generation of Activity and Sequence Diagrams from Natural Language Requirements". ENASE 2014: 69-77

[28] Paydar, S., Kahani, M. (2015) "A semi-automated approach to adapt activity diagrams for new use cases". Information & Software Technology 57: 543-570 (2015)