

Automated Check of Architectural Models Consistency using SPIN

Paola Inverardi, Henry Muccini & Patrizio Pelliccione
Dipartimento di Informatica
Università dell'Aquila - Via Vetoio, 1
67100 L'Aquila, Italy
{inverard, muccini, pellicci}@univaq.it

Abstract

In recent years the necessity to handle different aspects of the system separately has introduced the need to represent SA from different viewpoints. In particular, behavioral views are recognized to be one of the most attractive feature in the SA description and in practical contexts, state diagrams and scenarios are the most used tools to model this view. Although very expressive this approach has two drawbacks: system specification incompleteness and view consistency.

Our work can be put in this context with the aim to manage incompleteness and to check views conformance: we suppose to have state diagrams and scenarios models representing the system dynamics at the architectural level; they can be incomplete and we want to prove that they describe, from different viewpoints, the same system behavior. To reach this goal, we are using the SPIN model checker and we are implementing a tool to manage the translation of architectural models in Promela and LTL.

1. Introduction

Software Architectures are recognized to be a powerful tool to produce increasingly complex software with shorter time-to-market and better quality.

Software Architectures (SAs) emerged in the '90 to structure complex software systems and nowadays it is an autonomous discipline with its own tools and formalisms. It focuses on the overall organization of large software systems and it is typically used as a high level design blueprint of the system to be used during the system development and later on for maintenance and reuse.

More recently, researchers in industry and academia are putting SA in practice, learning some lessons: i) despite the high level of abstraction, notably in industrial contexts, the SA can be *too complex* to be described and managed; ii)

SA production and management is, in general, an expensive task.

To tackle system complexity several viewpoints [15, 8, 17] have been introduced. The use of views allows to handle different aspects of the system separately. Several slightly different views (logical, module, deployment and so on) are defined in different approaches [15, 8, 2]. All of them agree that *dynamic aspects* are orthogonal to all the views and they allow for analysis and validation of architectural choices. Several works are using these dynamic descriptions for testing [3, 21], analysis and model checking [1, 4].

Even if there is not a standardized way to represent SA dynamics, in the current industrial practice *State-based machines* and *scenarios* are the most common used tools to model behavioral aspects: state diagrams describe components behavior while scenarios identify how they interact. Although very expressive, this approach has two drawbacks with respect to analysis and validation.

The first one deals with system specification *model incompleteness*: notably in industrial contexts, these models are not completely specifiable or become practically intractable due to state explosion.

The second is a problem of *view consistency*: the use of several views facilitate the growing of inconsistency between views representing different aspects of the same system [18, 20]. State diagrams and scenarios provide alternative views of the system: these views are not independent and can erroneously specify contradictory or inconsistent behaviors.

The approach we present in the paper addresses the two above problems. First we put ourselves in a practical industrial setting, thus assuming to deal with multiple views SA descriptions. Second we only deal with dynamic views of SA descriptions. In this setting we propose a framework in which it is possible to obtain, starting from the (incomplete) dynamic views, an actual SA complete model in Promela. Then we validate the obtained model with respect to a set of selected scenarios.

The paper is organized as follows. In the next section

we summarize all the hypothesis our framework is based on. Section 3 details our approach, Section 4 illustrates the architecture of our framework. Section 5 cites related works while the last section summarizes our approach and presents future research goals.

This paper full version may be found in [11].

2. Setting the context

Let us now define our assumptions on what an SA description should provide in order to integrate in our framework. Our general assumption is that the architectural description consists of components, connectors and channels: a component is a computational or storage unit, a connector is a particular kind of component, responsible for coordination and a channel is the way components and connectors interact. Components dynamics is expressed by state machines, connectors may be explicitly modeled by scenarios and a set of scenarios describe how components and connectors interact.

In the following sections, we will use a standard CCS notation to describe state diagrams and a stereotyped UML [19] notation¹ to refer to scenarios: the half stick arrow, the filled solid arrow and the slanted arrows represent asynchronous, synchronous and deferred synchronous communication. The stick arrow represents the generic communication.

The SPIN model checker [23], defines the model by using the Promela language [23] while the properties to be checked are defined by using Linear-time Temporal Logic (LTL) formulae [23]. Promela is a verification modeling language. It introduces a notation to model processes that can communicate via synchronous and asynchronous channels. The key elements used in the language are processes (proctypes), message channels and variables.

In the following we will then assume to start from SA descriptions expressed in our notations. We want to stress that the notations we are using are general enough to easy map into.

3. The approach

Our approach is composed of three steps: in the former, architectural models, describing the components behavior, are translated into a Promela specification. This specification can be enriched by adding information obtained from the scenarios. In the second step scenarios are translated into LTL formulae. Finally the SPIN model-checker checks if the LTL formulae are verified on the obtained global model. In the following subsections we will summarize

¹each rectangular box represents a component, each arrow defines a communication line between the left and the right boxes

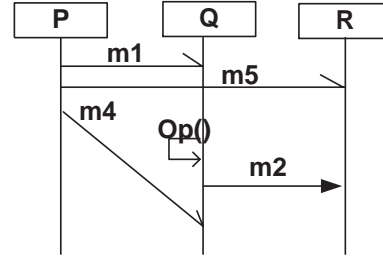


Figure 1. Sequence diagram example

the three steps, without giving too many details on how the translation algorithms work. The technical details may be found in [12].

3.1. Step one: generating the Promela model

In this step our goal is to generate a Promela specification of the architectural state diagrams. Architectural components are translated in Promela proctypes and their behaviors, modeled by the components state diagrams, are expressed through Promela statements. This kind of mapping, precisely described in [12], is similar to many others (as outlined in the related works) but its nature is mainly architectural. It means that, in our translation, we look at the architectural elements (components, connectors, channels, messages exchanged over the channels, and so on) and at the component state diagrams and translate them into Promela. Following this idea, the translation algorithm:

- i) maps component state diagrams into proctypes. Each proctype will model one state diagram behavior;
- ii) extracts connectors from scenarios and express these connectors by Promela proctypes;
- iii) maps architectural channels into Input and Output channels;
- iv) defines variables used during the SPIN execution to store information on *when* send and receive operations are performed over the defined channels. These variables will be used in Subsection 3.3 to analyze if the Promela model behavior conforms to the scenarios description.

3.2. Step two

This subsection explains how scenarios are translated into LTL formulae. Scenarios define a temporal ordered sequence of events, i.e., they define the order in which messages are sent and received through channels. Given a selected scenario, our algorithm analyzes the order in which arrows are defined in the scenario and the arrows type (synchronous, asynchronous or deferred) to identify *when* a send or a receive is performed over each channel.

Let's briefly describe an informal mapping between scenarios and temporal logic formulae by using the scenario in

Figure 1. The notation “ $Sm1 < Sm2$ ” stands for “ $m1$ is sent before $m2$ ”. Rm_i will be used to identify the reception of a message.

- Initially, P sends message $m1$ to Q using an asynchronous connector. It means that $Sm1 < Rm1$ but we cannot define precisely when $Rm1$ will be received. We suppose it will be received before the end of the scenario (END).
- P sends $m5$ to R in an asynchronous way. It means that $Sm1 < Sm5$ and $Sm1 < Rm1 < END$ and $Sm5 < Rm5 < END$;
- P sends $m4$ to Q using a deferred synchronous communication. It stands for $Sm1 < Sm5 < Sm4$ and $Sm1 < Rm1 < END$ and $Sm5 < Rm5 < END$ and $Sm4 < Rm2 < Rm4$. The last rule identifies the notion of “deferred” communication.
- Q synchronously sends $m2$ to R. The informal temporal logic formula now becomes $Sm1 < Sm5 < Sm4 < Rm2$ and $Sm1 < Rm1 < END$ and $Sm5 < Rm5 < END$ and $Sm4 < Rm2 < Rm4$.

3.3. Step three

The main goal of this approach is to check the conformance between scenarios and state diagrams. In other words, *we want to check if the temporal ordered sequence of events the scenarios describe are also exhibited by some paths in the complete model generated by SPIN.*

We run the global model obtained from the Promela specification and model check its execution with respect to the LTL formulae. When the Promela specification runs, the variables we described in our statediagram-to-Promela algorithm store information on *when* send and receive operations are performed over the defined channels. These information is compared (by the model checker) with the defined LTL formulae and behavioral errors trace are revealed.

Now there can be several ways of mapping the informal temporal formula, obtained in the previous subsection, into a LTL formula. Each one has slightly different semantics as detailed below.

We can run the model check operation with some options, each one corresponding to a different LTL formula:

i) we can check if there *exists* at least an architectural behavior that conforms to our informal formula or if *all* the architectural paths conform to the selected scenarios.

ii) we can check the formula imposing that the only actions allowed are those specified by the scenario (*strict check*) or other actions are allowed (*loose check*) as long as the fixed temporal ordering is respected. In the strict check, the formula is written imposing that $Sm1 < Sm2$ and $Sm1$ is

the first operation and $Sm2$ is the second. In the loose check, we only impose that $Sm1 < Sm2$ that implies that something may happen before or after $Sm1$.

This types of checks are embedded in the way the LTL formula is written. Different check options result in different analysis we are interested in. For example, the *exists & loose* options check that at least one state machine path “loosely” behaves as the selected scenario while the *all & strict* options verify that all the paths in the Promela model “strictly” behaves as the scenario.

4. The Charmy Tool: an easy way to practice the approach

To support our approach we are developing a tool, called Charmy (CHecking ARchitecture Models consistency), to help the software architect to perform the consistency checking of its architectural models. A prototype version is now available. It enables the software engineer to draw architectural state diagrams and scenarios (using the ScenariosEditor and StatechartsEditor components) and to automatically translate these models in Promela and LTL.

To experiment the approach in practice and to test the tool we analyzed the software architecture models of two different systems: the Teleservices and Remote Medical Care System (TRMCS) [9] and the Charmy tool SA itself. For lack of space, the TRMCS and the Charmy architectures, state diagrams and scenarios are fully described in [10] and [11] respectively. We will spend, here, only some words just to summarize some relevant results.

The TRMCS has been model-checked with respect to three scenarios of interest (in [10]): Scenario1 puts in evidence a low concurrency capacity while Scenario 3 highlights a possible system bottleneck.

The Charmy architecture has been model-checked with respect to four selected scenarios [11]. Particularly interesting are Scenario b and Scenario d in which storage problems and unexpected flows are pointed out, respectively.

5. Related works

The general themes of system (in)completeness and (in)consistency have received a lot of attention in the last few years, showing the growing interest on this area, see e.g. [14, 13].

As outlined in [5] consistency checking among multi-perspective or multi-language specifications is not a young field of research. There are different papers proposing different approaches to handle consistency.

Some of these are used for checking model consistency: in [6] the software views, and their relationships, are formally defined as sets of graphs and they are subsequently

checked for consistency; in [5] the authors present how inconsistencies may be handled using the *ViewPoints* framework and a logic-based consistency handling.

In [22] the Hugo tool is presented. It has been designed to automatically verify whether the interactions expressed by a collaboration diagram can indeed be realized by a set of state machines.

Another class of works that can be related to our is based on the idea of translating models in Promela or LTL and using SPIN to model check them. In [16] and [7] the authors translate UML State Diagrams in Promela and Hierarchical Automata, respectively. The model correctness is validated using SPIN and JACK, respectively.

6. Conclusions and future work

In this paper we presented an approach that, starting from a SA incomplete model, allows to generate a complete Promela model and model check it with respect to selected scenarios. The approach has been automated through the Charmy tool and two real case systems have been checked.

In ongoing works we are applying the approach in a project dealing with the Java two Enterprise Edition (J2EE) architecture to demonstrate some kind of “behavioral equivalences” between different architectural models.

Some ideas for future work are to consider enriched statecharts and scenarios to prove that also quantitative or temporal requirements are met by the specification and to prove UML statecharts and Sequence diagrams conformance, extending our algorithms.

As far as the tool is concerned, future work consider to extend the Charmy to cope with UML State diagrams and Sequence Diagrams formalisms and to be able to input automata in FC2, CADP or XML format.

Acknowledgments

The authors acknowledge the Italian M.U.R.S.T. national project SALADIN that partly supported this work and Leonardo Quaglia that developed the Charmy tool.

References

- [1] R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, July 1997.
- [2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley, SEI Series, 1998.
- [3] A. Bertolino, P. Inverardi, and H. Muccini. An Explorative Journey from Architectural Tests Definition down to Code Tests Execution. *On ACM Proc. ICSE2001*, June 2001.
- [4] D. Compare, P. Inverardi, and A. L. Wolf. Uncovering Architectural Mismatch in Dynamic Behavior. *Science of Computer Programming*, 33(2):101–131, February 1999.
- [5] A. Filkenstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multi-perspective specifications. *IEEE Trans. on Software Engineering*, 20(8):569–578, August 1994.
- [6] P. Fradet, D. Le Metayer, and M. Perin. Consistency Checking for Multiple View Software Architectures. *In Proc. European Software Engineering Conference (ESEC/FSE’99)*, pages 410–428, 1999.
- [7] S. Gnesi, D. Latella, and M. Massink. Model Checking UML Statecharts Diagrams using JACK. *Proc. Fourth IEEE International Symposium on High Assurance Systems Engineering*, IEEE Press, 1999.
- [8] C. Hofmeister, R. L. Nord, and D. Soni. *Applied Software Architecture*. Addison Wesley, 1999.
- [9] P. Inverardi and H. Muccini. The Teleservices and Remote Medical Care System (TRMCS). *Case study of the IWSSD-10 workshop*, San Diego, California, November 2000.
- [10] P. Inverardi, H. Muccini, and P. Pelliccione. Applying the Approach to the TRMCS case-study. On-line at: <http://www.dm.univaq.it/~muccini/Page2.html>.
- [11] P. Inverardi, H. Muccini, and P. Pelliccione. Automated Check of Architectural Models Consistency using SPIN. *IR 11/01, University of L’Aquila*, On-line at: <http://www.dm.univaq.it/~muccini/Page2.html>.
- [12] P. Inverardi, H. Muccini, and P. Pelliccione. Checking consistency between architectural models using SPIN. *IR 02/01, University of L’Aquila*, On-line at: <http://www.dm.univaq.it/~muccini/Page2.html>.
- [13] IWL. Second Int. Workshop on Living with Inconsistency. May 13, 2001 Toronto, Canada (Part of ICSE’01).
- [14] IWSSD2000. Tenth Int. Workshop on Software Specification and Design. *Track on (In)completeness, (in)consistency and (sufficient) correctness*, San Diego, November 2000.
- [15] P. Kruchten. Architectural Blueprints - The “4+1” View Model of Software Architecture. *IEEE Software*, 12(6), pages 42-50, November 1995.
- [16] J. Lilius and I. P. Paltor. Formalising UML State Machines for Model Checking. *In UML’99 - The Unified Modeling Language. Beyond the Standard*, Fort Collins, October 1999.
- [17] J. Magee, J. Kramer, and D. Giannakopoulou. Behaviour Analysis of Software Architectures. *In Proc. First Working IFIP Conference on Software Architecture (WICSA1)*, San Antonio, Texas, 1999.
- [18] C. Pons, R. Giandini, and G. Baum. Dependency Relations Between Models in the Unified Process. *In Proc. IWSSD 2000*, November 2000.
- [19] Rational-Corporation. Uml Resource Center. UML documentation, version 1.3. On-line at: <http://www.rational.com/uml/index.jhtml>.
- [20] G. Reggio, M. Cerioli, and E. Astesiano. Towards a Rigorous Semantics of UML Supporting its Multiview Approach. *In Proc. FASE 2001, LNCS n. 2029, Springer Verlag*, 2001.
- [21] D. J. Richardson and A. L. Wolf. Software Testing at the Architectural Level. *Second International Software Architecture Workshop (ISAW-2)*, October 1996.
- [22] T. Schafer, A. Knapp, and S. Merz. Model Checking UML State Machines and Collaborations. *On the Workshop on Software Model Checking*, July 2001. ENTCS, vol. 55 n. 3.
- [23] SPIN. On line at: <http://cm.bell-labs.com/cm/cs/what/spin/index.html>.