Reverse Engineering Models from Traces to Validate Distributed Systems – An Industrial Case Study

Andreas Ulrich[1] and Alexandre Petrenko[2]

[1]Siemens AG, Corporate Research & Technologies CT SE 1
Otto-Hahn-Ring 6, 81730 Munich, Germany
andreas.ulrich@siemens.com
[2]CRIM, 550 Sherbrooke West, Suite 100, Montreal, H3A 1B9, Canada
petrenko@crim.ca

Abstract. The paper targets the applicability of model-driven methodologies to the validation of complex systems and presents a case study of a mobile radio network. Validation relies on the availability of a collection of models formally describing various aspects of the system behavior and an execution trace obtained through monitoring the system during the execution of designated test cases. The models describe system properties and are derived from existing (informal) system specifications or other traces. The recorded trace is reverse-engineered to produce a model of the system that is used to visualize the architecture of the system during test execution and to verify the system against the specified properties using model checking technology. The obtained results and lessons learned from this case study are discussed.

Keywords: Model-driven development, reverse engineering, model verification, trace analysis, system validation, telecommunication industry, experience report.

1 Introduction

Model driven development (MDD) methodologies allow the separation of domain concerns from implementation details, which provides better quality, maintainability, and portability of the developed systems across different platforms. However, design models, which are essential to the success of the MDD approach, are not always sufficiently expressive to allow full automation of critical development activities, such as code generation. Moreover, models of complex systems, e.g., distributed applications, are seldom complete and, at best, cover only parts of the whole system. These limitations affect the applicability of MDD methodologies to system development. Nevertheless, partial models can still be used in reasoning about the developed system and its behavior. A (partial) model serves as a property that describes a singular aspect of the system. During validation, the violation or satisfaction of the property in an execution trace recorded during test execution of the system can help evaluate the functional correctness, the quality, and even the performance of the developed system.

Thus, there is a need for tools that rely on monitoring functions of distributed systems to produce log files of execution traces that can be analyzed further. This type of analysis is also known as runtime verification or passive testing of distributed systems

D.H. Akehurst, R. Vogel, and R.F. Paige (Eds.): ECMDA-FA 2007, LNCS 4530, pp. 184–193, 2007.

[2]. In this context, an approach has been developed [5, 6, 7] that takes as input an execution trace of the system obtained during the execution of a test scenario. Such a recorded trace, which contains a causally ordered sequence of send/receive events or messages exchanged between system components, is reverse-engineered to produce a model in the form of a system of communicating state machines that reflects the system behavior as it occurred during the execution of that particular test scenario. The obtained model is then used to verify the specified system properties using a standard model checker. The collection of verified system properties can later be used as a partial formal model of the system.

While the underlying theoretical aspects of such approaches to system validation are understood, much work remains to be done before the appropriate technology is widely accepted in industry. In particular, we believe it is important to conduct numerous industrial case studies to a) raise awareness about it, b) stimulate development of better tools, c) identify ways of making validation technology more lightweight, and d) identify directions for further research in the field. This paper reports on such an industrial case study.

The rest of the paper is organized as follows. The next section gives a short overview of the case study. Section 3 discusses the system validation approach using trace analysis. Next, Section 4 describes an example in more detail. Before the paper is concluded, Section 5 discusses and summarizes the results and experiences gained from this case study.

2 Case Study Overview

A model checking approach to validation is particularly useful when distributed systems allow an easy capturing of their internal communications. Such monitoring is often used in telecommunication networks for debugging purposes, relying often on protocol analyzers and network tracers. In the chosen industrial case study, we focus on the applicability of a model checking based validation technology we developed earlier [5] to end-to-end testing of a 3GPP UMTS radio network [11]. Our goal is not only to demonstrate to testers that MDD can be of help to them, but also to identify research direction for improving the applicability of this technology.

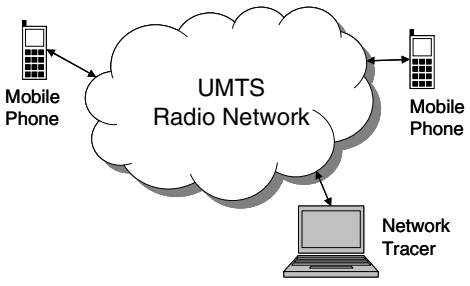

Fig. 1. Schematic overview of the UMTS radio network, which serves as SUT

When an entire 3GPP UMTS radio network is tested end-to-end, the result of a test run cannot be concluded alone from the behavior observed at the mobile phones (test probes) because of the high complexity of the network. Therefore, the communication at various internal interfaces is analyzed in addition to compute the final test verdict. In this system, communications are monitored in a non-intrusive way using designated tracing tools (network tracers) and are recorded in execution traces. Note that the whole UMTS radio network comprising several nodes (Node-Bs, RNCs etc.) serves as the system under test (SUT) as shown in Fig. 1.

3 Validation Using a Trace Analysis Approach

3.1 Outline of the Approach

The approach to validate the system based on observed traces is illustrated in Fig. 2 and can be summarized as follows:

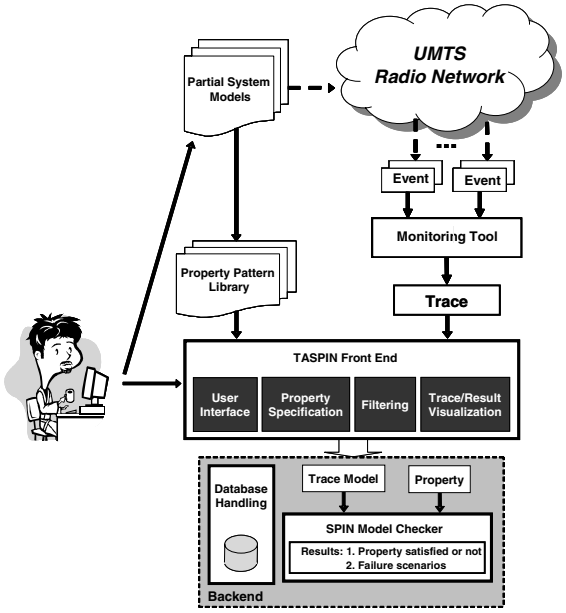

Fig. 2. Workflow of the modeling and system validation approach

- A trace during test case execution is captured that records the messages exchanged internally between different network nodes involved in the communication of two or more mobile phones.
- Traces are fed into the TASPIN front-end tool (Eclipse based) that converts the trace into a behavioral model (here: a Promela model that is input to the Spin model checker)

- Properties describing partial aspects of the system are specified in UML2 diagrams; currently sequence diagrams are used to express so-called base patterns of unique network features and activity diagrams to describe base pattern composition; a library of patterns is also built.
- Model verification: The reconstructed behavioral model (in Promela) plus the desired properties (UML2 diagrams converted to Promela Never Claim expressions) are fed into the Spin model checker backend.
- Verification results (failure traces) and the trace model itself are visualized as UML2 sequence diagrams.

The properties considered are derived from specifications of the 3GPP standards [1] and serve as a reference that must be matched in the recorded communication trace and ultimately by the SUT. They can be also considered as a repository of partial system models that are used to validate test scenarios. This repository of models will grow over time because more models are added during the course of testing and validation. The models assist not only the system validation; they are also of help when an MDD approach to develop new system features is next followed.

3.2 Implementation of the Approach

The workflow for system validation has been implemented in an Eclipse environment [3] using Spin as a model checker [8, 10] in the background. The implemented components are briefly explained (cf. Fig. 2).

- **Eclipse user interface:** Eclipse is used as the implementation environment for all the components of the tool. It comprises the following functionalities:
  - Loading and accessing traces and results in the database
  - Trace and result visualization
  - Filtering
  - Modeling in Promela
  - Property specification and selection
  - Controlling Spin
- **Database handling:** The heart of the module is the MySQL database [9] that manages all information relevant for system validation.
  - Saving/retrieving trace information into/from the database: The module takes UMTS traces recorded in XML format and stores them in the database.
  - Loading/retrieving analysis results into/from the database: Once the verification is done and the corresponding example/counter example is produced, the result file generated by the Spin model checker is read in order to retrieve the information in the original trace.
- **Filtering:** Filtering is implemented using SQL queries that select the desired messages from the database. So far there are three main categories of filters that can be combined to form user defined filters that respond to his needs and understanding of the trace:
  - Filtering by messages
  - Filtering by protocol layers
  - Filtering by parameters

- **Visualization of traces and analysis results:** This component is implemented using the Eclipse TPTP platform [4]. Its tracing and monitoring tools allow showing traces as lists of messages or as interactions between processes in a UML sequence diagram like representation. The same visualization mechanism is used to visualize both the trace and the analysis results read from output files generated by Spin.
- **Property specification:** The properties are obtained from an in-house UML editor that has been implemented as an Eclipse plug-in. The component also allows to select and instantiate a property to be considered for system validation. The detailed property specification procedure is described in Section 3.3.
- **Trace modeling:** The basic function for this module is to read the messages selected by the desired query from the database and write a model in Promela to represent them. There can be two different models produced:
  - Generate a model that does not take into account the concurrency within the SUT. In such a case, the model will consist of a single linear automaton that depicts the exchange of the recorded messages in the order of their appearance in the trace file.
  - Generate a model that features the communications between the processes and reflects the concurrent nature of the SUT. The tool generates the Promela model from a trace by defining a signal corresponding to each message and creating channels that carry the defined signals. The produced model in Spin is an asynchronous model, where processes communicate over unidirectional channels.

3.3 Property Specification

A pragmatic approach has been chosen to identify the common properties to be validated in UMTS traces. It mainly relies on the standardized 3GPP specifications of the network services that define the functionality of the UMTS network and deduces the patterns that make up the services and functions from them.

The main idea of the approach is to define so-called base patterns first and then to combine them to obtain more complex patterns. Base patterns describe a certain service taking place between the relevant network nodes. UML2 sequence diagrams (i.e., message sequence charts supporting alternative, parallel, iterative, optional and other behavior specifications) are used to specify the behavior in a base pattern. Typically the base patterns encode the behavior the system should normally exhibit (the expected behavior), but they can be also used to formulate undesired behavior, e.g., the occurrence of unexpected (error) messages.

Moreover, base patterns are parameterized by message data fields to make them applicable for other traces and re-usable for other system validation tasks. Classifying and storing base patterns in a library will foster re-use and reduce total validation efforts.

In a typical system validation task, the system engineer decides what kind of services the recorded trace of an executed test case should contain and in what order they should occur. She/he takes out the required base patterns from the library if available or creates them beforehand. In a next step, the base patterns are composed in a UML2 activity diagram to specify the expected ordering of their occurrence in the trace. Eventually the base patterns are instantiated replacing the formal parameters by actual values taken from the test run.

The following list describes the possible ways to combine base patterns:

- *Sequentially*: The last message of the first pattern occurs before the first message of the second pattern.
- *Concurrently*: The messages of the two patterns can interleave.
- *Alternatively*: If the messages of the first pattern appear in a certain trace segment, the messages of the second pattern shall not occur in the same segment.
- *Iteratively*: The messages of a pattern are repeated for a finite number of times.

Properties to be verified by Spin can be written in LTL or in *Never Claim* expressions [8]. For purposes of implementation and practicality, we have chosen Never Claim representations. First of all, LTL is more difficult to understand and to write for end users and even model-checking experts. Also it does not map well to the adopted UML diagrams.

The range of properties that can be verified in the Spin approach is defined by the expressiveness of the Never Claim expressions of the Promela language itself. In this project, we are concerned more about defining meaningful properties that help evaluate the system and offering a lightweight approach to model-checking that is acceptable by practitioners. Never Claims are sufficiently expressive for the properties considered in this project so far. Base patterns can be repeated in a trace several times and in various orders.

3.4 Using Patterns

Patterns can be devised and used for system validation following various strategies depending on, for example, the current level of confidence about system correctness. At the initial stage of testing, the system engineer may need to devise patterns from existing (informal) system specifications. If she/he already trusts several executions traces, she/he may want to make sure that certain patterns are repeated also in a newly obtained trace. Starting from a trace that is known to be correct and complete in terms of certain services and features, some base patterns can be identified and a *reference trace* (in terms of UML2 sequence and activity diagrams) is defined. The reference trace can then be used to check new traces. The traces that violate the reference trace are declared faulty unless they have been analyzed further and are known for sure to be correct, in which case the reference trace must be updated.

4 Example

The example discussed in this paper refers to a test case implementing a user scenario that tests the execution of a package-switched call between two mobile phones in order to transfer data from one mobile phone to the other one using the FTP command. Naturally, a number of network services at different network nodes and interfaces are involved in this user scenario.

In our trace-based approach to system validation we describe each required network service as a base pattern and combine them according to their temporal appearance and ordering in the user scenario as recorded in the trace. As an example of a base pattern, Fig. 3 shows the message flow of a so-called “RRC connection establishment” procedure that is issued each time a mobile phone attempts to establish a connection. Other base patterns are modeled in a similar way, but could contain more complex control structures such as parallel or alternative messages and loops.

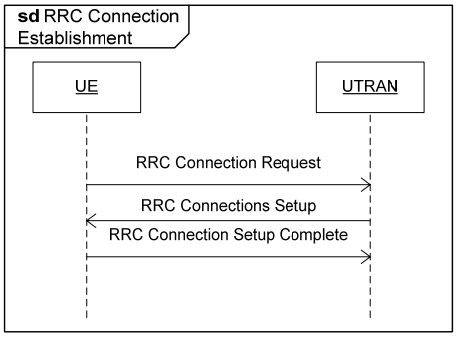

Fig. 3. Base pattern "RRC Connection Establishment"

Next, a set of base patterns is combined in a composite pattern (Fig. 4) to reflect the expected behavior of the user scenario in the trace. The UML activity diagram uses different swim lanes for each service group, here RRC and GPRS services. This kind of structuring improves the readability of the diagram in particular. The composite pattern in Fig. 4 can be also viewed as a reference trace that must reoccur each time the associated test case is performed.

A recorded trace in this example contains several hundred messages that occurred at various interfaces of the radio network during test case execution. Only about 30 messages are relevant because they are part of one or another base pattern that needs to be verified. These relevant messages could be filtered out from the trace prior to generating a Promela model. Otherwise the model is unnecessarily cluttered with unneeded messages that only slow down the model verification. At the moment, filtering is a manual process. However, we are currently discussing an approach to automatically filter out messages from the trace based on the selected patterns to be verified.

The generated Promela model of the recorded trace and the composite pattern of the expected behavior translated into a Never Claim expression are eventually fed into the Spin model checker. In the simplest case a linear Promela model is used, which does not pose much stress on the model checker. Even when using a concurrency model for the trace (see Section 3.2), the approach remains scalable if the relevant messages were filtered out before model creation.

The result of the model-checking task is a simple statement whether the verified property is contained in the model or not. In our context it means that we get assurance whether the composite pattern is matched by the recorded trace or not. In the negative case a failure scenario is produced by Spin that depicts the path from the initial state of the model, i.e., from the first message in the trace, to the state, at which diverging behavior is detected. The failure scenario is mapped back to the original trace by our tool such that the system engineer gets hints, which message violated the specified behavior of the provided composite pattern.

System engineers at Siemens used the described approach mainly during regression testing of radio network elements to validate the correct execution of user scenarios. Their validation task becomes now much simpler since before they had to analyze the produced large trace files manually in a text-based trace viewer. By means of our tool they

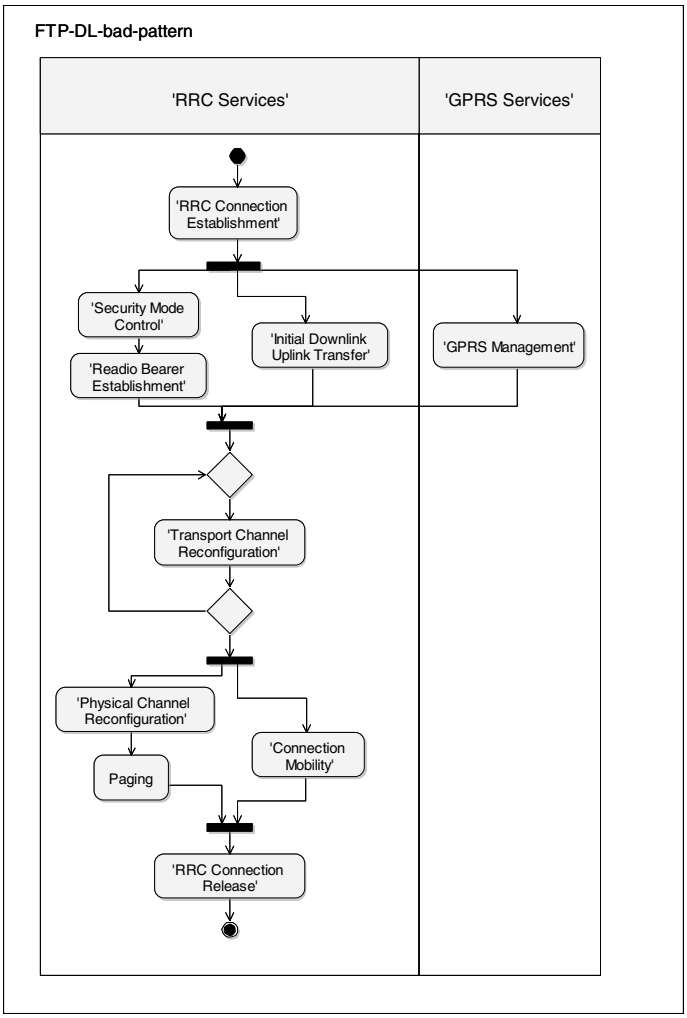

Fig. 4. Composite pattern of the expected trace behavior

are enabled to quickly decide about the correctness of network services. The graphical representation of base patterns and composite patterns is in particular helpful for them to understand the validation results and discuss them with other project members.

5 Results and Experiences Obtained

The presented approach to system validation is based on an analysis of traces recorded during the execution of test cases or by some other means. Therefore it is particularly well suited for poorly documented systems. Our approach produces a set of partial system models (base patterns and compositions) that are used to verify the

correct behavior of services and other system features. They can be considered also as a starting point to reverse engineer (parts of) the system if it is desirable, e.g., for maintenance reasons.

With our trace analysis approach we also attempt to make sophisticated solutions like model-checking techniques available to practitioners who are not well trained in theoretical computer science at all. Up to now, formal verification is still an area that receives little attention in practice. We believe that this promising area can only succeed if the entry hurdle of this technology is sufficiently low. This refers in particular to the way how system properties are specified. UML seems to be again an appropriate mean for this purpose. Although it might have a smaller expressiveness compared to temporal logic languages that are commonly used in model checking, its graphical format and easy readability are of major benefit.

System models are modeled in terms of base patterns and composite patterns. This kind of abstraction works well for the discussed telecommunication domain. It might be interesting to see how this concept could be transferred to other application domains as well. Nevertheless, the mapping from UML2 sequence diagrams and activity diagrams to Promela Never Claim expressions as used in our approach is still preliminary since the semantics on the composition of base patterns in particular requires further research. While simple compositions are easily supported by our tool, our experience in specifying complex patterns indicates that there is a need to define a formal *language for pattern specification*.

In this case study it can be concluded that the chosen system validation approach based on reengineering models from traces is feasible and produces valuable results. At the same time, the integration of an UML editor used to specify system properties with an off-the-shelf model-checker is still a daunting task. So far, UML behavioral diagrams such as sequence diagrams and activity diagrams have to be transformed into an appropriate input of the model-checker. It would be desirable that future modeling tools would integrate model-checking techniques on their own such that model validation can be done more easily.

6 Conclusions

We used in our case study a model-checker approach for system validation based on traces recorded during test execution of a UMTS radio network acting as the system under test. The tool has been implemented within Eclipse and integrates with a UML editor used for the specification of properties and Spin as the model-checker. The main open problems are the completion of an extensive library of base patterns for properties in UML and a refinement of the integration with Spin. Such a refinement should result in an improved user interface to the verification task.

We believe that the presented approach to system validation is also applicable to other types of distributed systems provided that expressive traces during system runtime can be obtained. Besides applying the approach during end-to-end system testing, it could also be used in an uncontrollable environment, e.g. during field testing. Furthermore, reverse-engineering of traces seems the only feasible way to deal with systems that lack sufficient and up-to-date design documentation.

Acknowledgement

Crucial contributions of Hesham Hallal and ElHachemi Alikacem (CRIM) to tool development and support are acknowledged.

References

1. 3GPP Specifications Home; accessed, 2007-03-29 (2007) http://www.3gpp.org/specs/specs.htm
2. Colin, S., Mariani, L.: Run-Time Verification. In: Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (eds.) Model-Based Testing of Reactive Systems. LNCS, vol. 3472, pp. 525–556. Springer, Heidelberg, ISBN 978-3-540-26278-7 (2005)
3. Eclipse – An Open Development Platform; accessed 2007-03-29, (2007) http://www.eclipse.org/
4. Eclipse Test and Performance Tools Platform (TPTP) Project; accessed 2007-03-29, (2007) http://www.eclipse.org/tptp/
5. Hallal, H.H., Boroday, S., Petrenko, A., Ulrich, A.: A formal approach to property testing in causally consistent distributed traces. Formal Aspects of Computing 18(1), 63–83 ISSN 0934-5043 (2006)
6. Hallal, H., Boroday, S., Ulrich, A., Petrenko, A.: An Automata-based Approach to Property Testing in Event Traces. In: Proc. of the IFIP TC6/WG6.1 XV International Conference on Testing of Communicating Systems (TestCom 2003), pp. 180-196. Sophia Antipolis, France (May 2003)
7. Hallal, H., Petrenko, A., Ulrich, A., Boroday, S.: Using SDL Tools to Test Properties of Distributed Systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, Springer, Heidelberg (2001)
8. Holzmann, G.J.: The SPIN Model Checker; Addison-Wesley; ISBN. Addison-Wesley, London, UK, ISBN 978-0-321-22862-8 (2004)
9. MySQL – The world's most popular open source database; accessed 2007-03-29, (2007) http://www.mysql.com/
10. On-the-fly, LTL Model Checking with SPIN; accessed 2007-03-29, (2007) http://spinroot.com/spin/whatispin.html
11. Sauter, M.: Communication Systems for the Mobile Information Society. John Wiley, New York, ISBN 978-0-470-02676-2 (2006)