

TOWARDS A UML PROFILE FOR TRACE ANALYSIS OF DISTRIBUTED SYSTEMS

by

Hesham Hallal^{1,3}, Alexandre Petrenko¹, Sergiy Boroday¹, Andreas Ulrich²

¹ CRIM, 550 Sherbrooke West, Suite 100, Montreal, H3A 1B9, Canada,
{Hallal, Boroday, Petrenko}@crim.ca

² Siemens AG, Corporate Technology CT SE1, Otto-Hahn-Ring 6, 81730 München, Germany,
andreas.ulrich@siemens.com

³ Department of Electrical and Computer Engineering, McGill University, Montreal, Canada

Prepared for the 2005 Telelogic Americas User Group Conference

Abstract

TOWARDS A UML PROFILE FOR TRACE ANALYSIS OF DISTRIBUTED SYSTEMS

UML concepts and tools are increasingly applied in several areas of software engineering. The recent additions in UML 2.0 and the supporting tools, especially Tau G2, open the door even for wider applicability of these concepts and tools in areas like distributed system development. In this paper, we report on the ongoing work to define a UML profile in Tau G2 for the Trace Analysis of Distributed Systems. The proposed profile is based on the existing Formal Trace Analysis Framework, which uses traces of distributed systems to produce formal models (Currently in SDL) approximating the behavior of the system under test and allows the verification of the resulting models against properties specified in the GOAL language of ObjectGEODE. We believe that the Trace Analysis (TA) UML profile would allow the users of the trace analysis framework to benefit from the variety of modeling tools available in UML.

Author Biography

Hesham Hallal^{1,3}, Alexandre Petrenko¹, Sergiy Boroday¹, Andreas Ulrich²

Hesham Hallal has a Master's degree in electrical and computer engineering. He is currently pursuing his Ph.D. degree in electrical and computer engineering at McGill University. His Ph.D. studies focus on the topic of testing and controlling distributed systems using formal methods. He has four years of engineering experience and five years of research and development experience. Currently he is a senior research officer at CRIM.

Alexandre Petrenko, Ph.D. in computer science, is a senior researcher and team leader of Distributed System Analysis Group at CRIM (Computer Research Institut of Montreal). He sits on program committees of several international conferences such as TESTCOM, FORTE, ICNP, and FATES. He also gives invited talks at the main research institutes around the world. Alexandre has published over 150 papers and supervised many master and

Ph.D. students. His research interests include formal methods and their applications in the distributed systems and computer networks.

Sergiy Boroday has a Ph.D. in testing theory. He has been working in research and development for more than nine years. Currently he is a researcher at CRIM. He is winner of the 3rd degree diploma at the Intl. Conference "Students for Peace and Scientific Progress" (Russia, 1992) and the best paper award in the FORTE/PSTV'99 (China, 1999). His research interests include formal methods, automata theory, testing theory and methodology, telecommunication protocols.

Andreas Ulrich received his Ph.D. in Computer Science from Magdeburg University, Germany in 1998. He then joined Siemens Corporate Technology Division, where he works as a Principal Engineer in the Department of Software & Engineering until now. His main task is to provide consultancy to Siemens' business units in the area of testing and quality assurance. He is also active in the research area of model-based system analysis, verification, and testing.

¹ CRIM, 550 Sherbrooke West, Suite 100, Montreal, H3A 1B9, Canada,
{Hallal, Boroday, Petrenko}@crim.ca
Tel: 1- 514-840-1234

² Siemens AG, Corporate Technology CT SE1, Otto-Hahn-Ring 6, 81730 München, Germany,
andreas.ulrich@siemens.com

³ Department of Electrical and Computer Engineering, McGill University, Montreal, Canada

SCENARIO

The analysis of distributed systems is a complex task and relies heavily on automation. In recent years, trace analysis of distributed applications has become a reliable and common approach to analyze and evaluate the behavior of an application that consists of several processes running on different sites. This approach has also been enhanced by the growing use of formal methods in activities such as verification and testing. The main idea behind a trace analysis approach that uses formal methods is to collect traces (logfiles of events or messages) from the execution of the system under test (SUT) through monitoring and to build behavioral models of the collected traces to test them against user defined properties. An efficient technique to test properties on the models of traces is model checking, which relies on the generation and exhaustive exploration of the reachable state space of the model. Once a trace of an SUT and the properties of interest are modeled in the languages of the chosen model checker and verified to determine the satisfaction of the property by the model. The results of the verification include a verdict whether the property was met in the model of the trace and examples to show the execution sequences that satisfied/violated the property. Such counter examples can be used to track errors to their origins and are usually helpful in fixing or improving the tested systems. A typical workflow of the trace analysis approach is depicted in Figure 1.

In a previous works [HBU03, HPU01], we presented the implementation of the trace analysis approach in the context of the model checker ObjectGEODE (OG) from Telelogic, where the traces were modeled using the Specification and Description Language (SDL) and the properties specified in the GOAL language of OG. The implementation included a front end tool to generate SDL models from collected traces, a library of parameterized property patterns encoding common properties in the area distributed systems, and a GUI to facilitate the use of the library and the instantiation of the patterns.

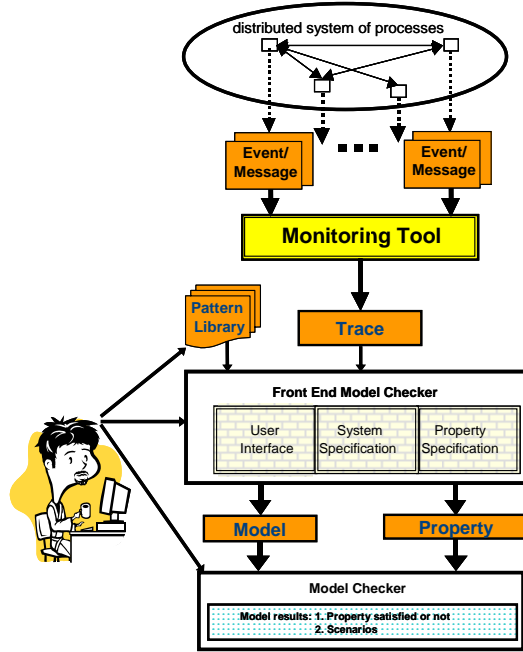


Figure 1. Trace Analysis workflow.

OBJECTIVE/PROBLEM

Recently, especially with the abortion of any further development of OG by Telelogic, we have been working on finding model checking alternatives to OG. At the same time, we are interested in enhancing the trace analysis framework so that it becomes more accessible and efficient. While model checking can be still performed by many (Open Source) available tools like SPIN [Hol97], SMV [Smv], and NuSMV [Nsmv], we face the challenge to find modeling and property specification alternatives to both SDL and GOAL while preserving their graphical representation and expressiveness, respectively. Using SDL provided us with standardized powerful language that suits the representation of traces for analysis in both textual and graphical representations suitable for processing and visualization of models. Meanwhile the automata based GOAL language offered us with a mean to express a wide variety of interesting properties. We actually built a library of parameterized GOAL observers to encode common properties, which facilitated the use of our trace analysis approach. Therefore, for any improvement on the current trace analysis framework, one should consider mainly these factors. In addition, the need to disseminate the technology to a wide clientele

necessitates the adoption of contemporary tools and means to model traces, analyze them, and interpret the results.

Meanwhile the emergence of the Unified Modeling Language (UML) as the standard modeling technique in the worlds of software and hardware, and the availability of many tools that support a UML modeling approach have made UML the target language for many applications in the area of system design and development. UML offers several notations which serve many of the tasks in the development lifecycle. For example, use case diagrams can be used to express the requirements of a system while state charts and activity diagrams can be used to model the behavior of the entities of the system (though on different levels of abstractions).

At the same time, there have been several attempts to devise techniques to analyze UML models in a formal setting. So, works like [GMP02] and [OGO04] describe how to translate the behavioral models written in UML into the input language of a model checker, where verification of (mainly) temporal properties could be preformed. In [OGO04], the authors even describe a framework which allows expressing properties in UML by introducing the notion of a UML observer. This framework provides also an environment to analyze UML models without resorting to a third party model checker.

For these reasons, we consider using UML as the main modeling option in the trace analysis framework. We plan to use UML to achieve the following:

1. Modeling traces. For this we need to use the UML notations that can reflect the behavioral description of the SUT as well as its structure. This includes state chart diagrams, architecture diagrams, and even activity diagrams.
2. Specifying properties. UML notations like use case diagrams and sequence diagrams have long been used to describe requirements in the software development cycle. For the trace analysis framework, we need to use a notation that can be easily used to verify the trace model against the property in concern. The options include the state chart diagrams similar to the work in [OGO04] where the UML observer notation is based on the state chart diagram concept. Other options include using activity diagrams and sequence diagrams for property specification.
3. Representation of the analysis results. It is usually preferable that the results be represented in the same notation as the traces in order to

guarantee tractability of the analysis results and to facilitate their interpretations.

In the following section, we discuss the preliminary results in our attempt to use UML, within the Tau tool from Telelogic [Tau], as the modeling language in trace analysis of distributed systems and the possibility of defining the UML profile for trace analysis.

SOLUTION

In this section, we discuss the preliminary results we obtained in our attempt to define a UML profile for the trace analysis of distributed systems. First, however, we present some of the features offered in the Tau tool that we used to represent traces and properties of distributed systems in the context of trace analysis. As discussed in the previous section, we consider the following aspects:

A. Trace Visualization

The visualization of a trace from the different perspectives includes the following main views:

1. Signal view : This is a class diagram containing the signals exchanged in the trace. There are two options to model signals:
 - a. Signal classes, where each signal is represented by a separate class with the parameters as its attributes.
 - b. Interfaces, where each interface class represents a set of signals that are exchanged in the system.
2. Static structure: this is a class diagrams that shows the processes of the trace as independent active classes along with the communication ports and the corresponding signals (interfaces).
3. Architecture plan: This diagram shows the interaction between the processes of the trace. It presents the communication ports and channels of each process.
4. Individual process behavioral view. The behavior of each process can be displayed using one or more UML diagrams in Tau. One can choose to show the behavior as an activity diagram or as a statechart diagram. In our case, we use the state chart diagrams so that we preserve the similarity to

SDL state machines. The UML statecharts are quite similar to the SDL state machines in that they allow using the same symbols for outputs and inputs and for tasks. The only difference is in using the parameters for signals (messages). In UML state charts, a parameter is defined in the signal and can be used directly for the output symbol in a statechart. However, for a process to receive a parameterized signal, the parameter must be defined in the receiving process as a local variable that needs to be assigned the value of the parameter.

B. Property Specification

As we mentioned earlier, requirements can be specified using several notations in UML, depending on the abstraction level. In our experiments, we considered using state charts and activity diagrams to model properties of distributed systems to be tested on traces. The choice is motivated by the fact that the properties that can be tested on traces express usually behavioral requirements, which means they should be modeled using notations that provide this aspect. A property can be represented similar to a process of trace, i.e., as an active class that has a behavior description.

C. Result Representation

When trace models are checked against properties, the model checker produces examples (counter examples) to show how the property was verified (violated) in the model. Therefore, the visualization of these examples (counter examples) is important to help interpret the analysis results. We use the sequence diagrams in UML to represent the results.

UML Profile for Trace Analysis

Figure 2 shows the hierarchical representation view. This view could be obtained using a class diagram that represents the entities in a trace analysis framework. A system consists of a trace and the properties to be verified on it. The trace in its turn can be decomposed into several processes. As for the properties, they can be represented as a system of properties that consists of the desired patterns or as a template of patterns that can be inherited by the desired properties. In fact one can consider Figure 2 as a preliminary overview of the components of the proposed trace analysis profile in UML. It does not show the component of the analysis results. A trace analysis system includes a trace and a several properties. The trace is composed of several processes (here we show five for illustration).

The properties are stored as patterns that can be parameterized. In addition, the profile includes several main components like:

1. Events: For traces that are collected by instrumentation, the basic entity is an event which can be a send, a receive, a rendezvous, or a local event.
2. Messages: When traces are collected by interception of communications between processes, the basic entity is a message that can be an asynchronous message or a rendezvous. The message is a generalization of the signal class in UML, where one can specify the source, destination, and timestamp.
3. Channels.
4. Predicates: Each property pattern consists of a collection of predicates that are combined by conjunction or disjunction. The basic parameterized expression of the predicate can be stored and the instantiations of several predicates combined by disjunction or conjunction makes the property.
5. Scenario: It represents the results from a verification process. Of a certain property. The scenario includes the examples or counter examples produced by the model checker. They are visualized as sequence diagrams. In addition, the scenario has a relation with the processes, events, and messages of the traces to indicate the entities that are involved in the verification of the property.

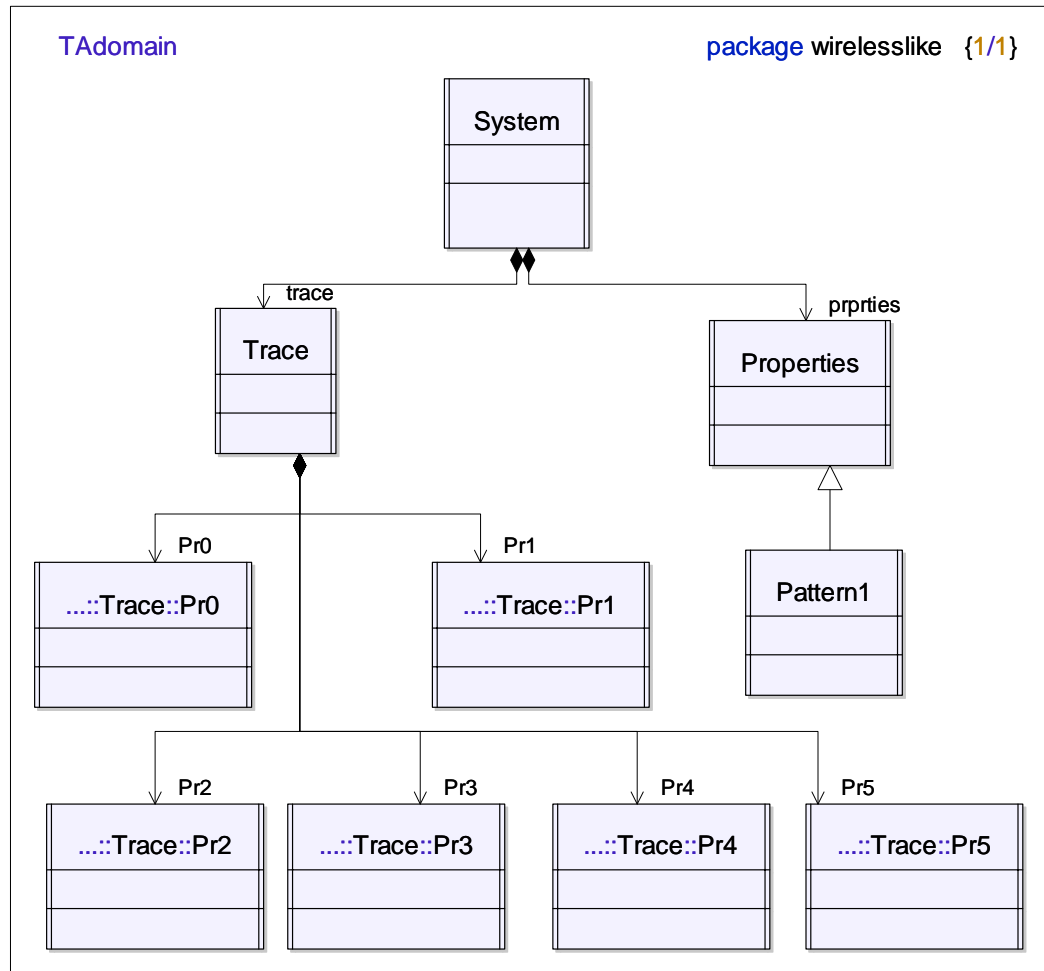


Figure 2. A hierarchical view of the trace and a desired property

RESULTS

This work in progress reveals that the creation of a profile of trace analysis of distributed systems in UML can bring many benefits to the dissemination of the technology, especially in the community of UML users. The preliminary results show that the task can be achieved with relative ease in Telelogic Tau, where the creation of new profiles is a well-documented straight forward procedure.

RESOURCES

- [GMP02] María del Mar Gallardo, Pedro Merino, Ernesto Pimentel. Debugging UML Designs with Model Checking. in Journal of Object Technology, vol. 1, no. 2, July{August 2002, pages 101{117, <http://www.jot.fm/issues/issue-2002-07/article1>
- [HBU03] H. Hallal, S. Boroday, A. Ulrich, and A. Petrenko. 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.
- [Hol97] G. J. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23(5): pp. 279-295, 1997.
- [HPU01] H. Hallal, A. Petrenko, A. Ulrich, S. Boroday. Using SDL Tools to Test Properties of Distributed Systems. In *Proc. of the Workshop on Formal Approaches to Testing of Software (FATES) in affiliation with CONCUR'01*; BRICS Technical Report NS-01-4, Aalborg, Denmark, August 2001.
- [Nsmv] IRST. nuSMV Home Page <http://nusmv.irst.itc.it/>
- [OGO04] Iulian Ober, Susanne Graf, Ileana Ober. Validation of UML models via a mapping to communicating extended timed automata. SPIN Workshop, 2004.
- [Smv] The SMV System, <http://www-2.cs.cmu.edu/~modelcheck/smv.html>
- [Tau] Telelogic. Tau. <http://www.telelogic.com/products/tau/>