Recent Extensions to Traviando

Peter Kemper
Department of Computer Science
College of William and Mary
Williamsburg, VA, USA
Email: kemper@cs.wm.edu

***Abstract*—Traviando is a trace analyzer and visualizer for simulation traces of discrete event dynamic systems. In this paper, we briefly outline recent extensions of Traviando towards an identification of model invariants and a detection of partial deadlocks and immediate events. This new functionality complements an existing model-checker and event browser. In addition to the graphical user interface, a new command-line version of the tool allows a user to obtain analysis results being documented in a set of generated web pages, which minimizes the learning curve for the use of Traviando to obtain some feedback on the details of a simulation run.**

***Keywords*-trace analysis; simulation; software tool**

I. Motivation

A model-based evaluation of systems or system design is made possible by powerful modeling environments. A subsequent analysis is often based on simulation. For a discrete event simulation, a simulation engine generates a sequence of events that is used to perform some statistical analysis or to drive an animation of the dynamic behavior. The value of the outcome of a simulation study relies on the model being valid, i.e., in close enough correspondence to the real system with respect to the measures of interest, and on the correct implementation of the model. Checking the latter is a step that is called model verification. Traviando supports this verification step by extracting model characteristics and by visualizing detailed trace data with message sequence charts. Traviando is designed to help a modeler find errors in a simulation run; it cannot help to prove the absence of errors for a given model. Traviando applies to discrete event systems in general and is not limited to a particular modeling formalism. It takes as input a sequence of states and events obtained from a simulation run and helps a modeler in recognizing critical behavior. It has been successfully used with various simulation frameworks, e.g., Möbius [2] directly exports trace data for Traviando, while the network simulator NS2 requires a separate trace formatter that reformats NS2 Nam traces accordingly [7]. Traviando was originally designed with a graphical user interface that offers a selection of features in pull-down menus. Those features included an event-brower [7], a model-checker, and trace reduction techniques [4]. In a recent extension, new functionality has been added that generates a report as a set of HTML formatted files. The generation of this report is possible by using command-line parameters. Figure 1 outlines the architecture: the input trace has a particular XML format (see [1] for a DTD), the tool itself is implemented in Java and uses several libraries like one supporting a SAX parser, jfreechart for graphics, and commons math library for statistics. The generated webpages contain links to static webpages with further explanations for the generated list of warnings. In this paper, we summarize recently added features for this report functionality which is described in more detail in [5].

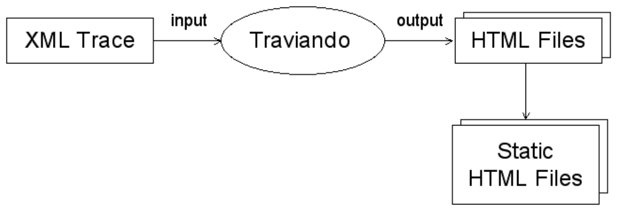

Figure 1. Commandline version of Traviando

II. Documenting observed behavior

One purpose of the generated report document is to summarize what information can be automatically extracted from the given trace data. The trace mainly provides events with time stamps and value settings for state variables. State variables can be grouped into sets named processes. The report starts with an overview page that outlines for each process how many variables are associated with it, how many actions perform only changes to variables of a single process (local actions), and how many interactions access variables of a process. The outline also contains figures for the number of value assignments to each variable and the number of occurrences for each action. It provides a figure that shows for each event if the simulation run reaches a new state or returns to a previous one and also a figure for the progress measure as defined in [4]. Similar outlines are generated for individual processes such that, for instance, the visual identification of patterns for the progress measure as suggested in [4] is straightforward.

For state variables, Traviando takes values of variables as strings by default and then recognizes integer and floating point variables. For each variable, Traviando reports its type and for numerical variables it reports the observed range of values, whether the sequence of values is monotonously increasing or decreasing, and characteristics like the mean value, variance and alike. It reports on which action makes changes to a variable and checks if the state transformation can be matched with a simple linear transformation $v' = c \cdot v + b$ where v' denotes the new value, v the old value and c, b are numerical constants calculated from observations. If the transformation does not fall into this category, Traviando reports on the number of assignments that increase (decrease) the value of a variable and on the most extreme changes observed. Pages with state variable information are linked with those for actions. For each action, the report contains its occurrence statistics and state transformations it performs to state variables. Traviando computes a calculation of state variable and action invariants with respect to numerical state variables, which uses related concepts in Petri net theory, for details see [6].

The general idea of the report is to have an easy-to-read, easy-to-navigate web report that helps a modeler to confirm expectations and to recognize obscure and unexpected behavior. For example, simple errors based on exceeding the expected numerical ranges of variables and erroneous state transformations performed by actions. Traviando's website [1] contains an example section with traces and generated reports.

III. Extract warnings

Traviando also generates a list of warnings that contains links to static web pages which explain the underlying rationale for each warning, indicate potential errors in the simulation and give guidance on how to verify and fix it. The approach is stimulated by the way software development is supported by static code analysis, where a software tool generates a report for a given program code that lists locations in the code base that require further attention based on a set of rules that incorporate knowledge on common pitfalls in programming; see FindBugs as a particular example [3]. Traviando attempts to accumulate knowledge on common errors and pitfalls in simulation modeling. For instance:

Simultaneous Events. A simulation may reach a time point where multiple events need to be scheduled. If this is the case, the order in which events are performed may be critical for the correctness a model. Since this is a very special case, it is often difficult to clarify how a particular simulator schedules events in those cases from either documentation or results produced by a simulator. If such situations are reached in a simulation at all, which actions are involved, and in which order, is reported by Traviando and given a corresponding warning.

Overflow of Variables. If enabling conditions of actions are not appropriately specified, actions may accidently overflow a value range for a numerical variable. Traviando assumes that if an observed range of values with lower bound l and upper bound u is within the range of a common numerical type like unsigned short or integer then this is the corresponding type. If it observes state transformations $v' = v + c$ such that $l + c$ or $u + c$ would exceed the assumed type then it creates a corresponding warning.

Deadlocks. A discrete event system model may be subject to partial or total deadlocks. While the latter is simple for a simulator to identify, the former is much harder to detect. Traviando checks certain conditions that indicate a potential deadlock. A condition for a variable is that it is not changed for a substantial portion of the trace towards the end of the trace and that it reaches an extremum in its range of values. The latter is based on the observation that models where entities allocate finite resources, a deadlock documents itself with state variables that describe the availability of resources being stuck at a value for resource exhaustion. A second condition is that actions that have been frequently seen throughout the trace are not seen towards the end of the trace. Obviously these observations do not prove presence of a deadlock and thus only create warnings.

The list of warnings gives guidance to a user who is looking for ideas what may cause a model to be defective.

IV. Summary

We outlined recently added functionality to Traviando, a software tool to analyze and visualize a simulation trace. Potential users are modelers who seek support for the debugging and verification of a simulation model as well as developers of simulation engines who want to debug and test simulator code. Traviando is available on request and free of charge for research and education purposes, for more information see [1].

References

[1] Traviando: www.cs.wm.edu/∼kemper/traviando.html, www.cs.wm.edu/∼kemper/traviando/examples.html.

[2] D. D. Deavours, et al. The Möbius framework and its implementation. *IEEE TSE*, 28(10):956–969, 2002.

[3] D. Hovemeyer and W. Pugh. Finding more null pointer bugs, but not too many. In *Proc. 7th Wshop Program Analysis for Software Tools and Engineering*, 2007.

[4] P. Kemper and C. Tepper. Automated trace analysis of discrete event system models. *IEEE TSE*, 35(2), 195-208, 2009.

[5] Peter Kemper. Report generation for simulation traces with Traviando. In *Proc. Dependable Systems and Networks*. IEEE CS, 2009.

[6] Peter Kemper. Recovering model invariants from simulation traces with Petri net analysis techniques. In *Proc. Winter Sim. Conf.*, ACM, 2009.

[7] Nathan J. Schmidt and Peter Kemper. Phrase based browsing for simulation traces of network protocols. In *Proc. Winter Sim. Conf.*, 2811–2819. ACM, 2008.