Distrib. Comput. (2012) 25:3–4
DOI 10.1007/s00446-011-0156-2

Concurrency theory: timed automata, testing, program synthesis

Davide Sangiorgi

Published online: 21 December 2011

Today, with the widespread use of Internet and with the increase in the complexity of the software, it becomes vital to develop methods for ensuring the quality of concurrent software systems. It is therefore natural that this area, broadly referred to as *concurrency theory*, be represented in the journal Distributed Computing. As a way of reassessing this relevance, the present issue of the journal is entirely devoted to concurrency theory. The four papers presented cover three important topics within concurrency theory: timed automata, automated program synthesis, and testing. I briefly comment on the contributions below.

Traditional automata model the interactions between a system and its environment by retaining only the sequencing of events. The behaviour of a system is thus the tree, or the set of sequences, of such events. The strength of automata theory lies in the efficient algorithms and decision procedures for automatically manipulating and analyzing the resulting behaviors. Many verification theories, including model checking, are based on the theory of automata, in one of their various variants (deterministic, Meely machines, Buchi, pushdown, etc.). While the abstraction from quantitative time has had many advantages, it prevents the applicability of the techniques to cases in which real-time constraints are essential. Timed automata were introduced by Alur and Dill in 1994 as a formal notation to model the behavior of real-time systems. In its definition, state-transitions are annotated with timing constraints using finitely many real-valued clock variables. Timed automata accept timed words; these are infinite sequences in which each symbol is associated with a real-valued time information. Stopwatch automata are a variant of timed automata where the clock variables can be set to on or off depending on the state. The main interest in this variant is its high expressive power. Today, time automata and their stopwatch extension are one of the most studied models for real-time systems. They are the base for a variety of tools for specification and verification of real-time systems, applied to the analysis of schedulability, asynchronous circuits, communication protocols and so on.

The paper "A study on shuffle, stopwatches and independently evolving clocks", by Catalin Dima and Ruggero Lanotte, makes a comprehensive study of properties of timed languages. The focus is the equivalence between automata and regular expressions, as in Kleene theorem, but in the setting of timed systems. The authors first show the equivalence between languages accepted by stopwatch automata and timed shuffle expressions (i.e., timed regular expressions with shuffle operator). Then, they restrict the class of stopwatch automata, so to obtain decidability of the emptiness problem. In this class, called partitioned stopwatch automata, the set of stopwatches is partitioned into disjoint classes. The class turns out to be equivalent with that of shuffle expressions with a certain fairness condition. Further, a correspondence is proved with the new class of distributed time-asynchronous automata: asynchronous compositions of timed automata in which time is allowed to progress independently between components.

In testing validation, tests are developed from a specification of the desired behavior, and then manually or automatically executed. The outcome of a test suite leads to a verdict about the correctness of the implementation with respect to the specification. Validation of distributed systems requires checking the kind of messages exchanged between individual components and the order in which they are exchanged. Typically, each component has an interface, consisting of

D. Sangiorgi (✉)
University of Bologna, Bologna, Italy
e-mail: Davide.Sangiorgi@cs.unibo.it

D. Sangiorgi
INRIA, Paris, France

various separate ports. When applying testing methods, one has to face a number of problems, including

- the concurrent execution of events from separate components or even from separate ports of the same component;
- various forms of non-determinisms;
- the absence of a global clock.

Assume that we place separate testers at each port of a system. This may require the need of synchronization between the local testers. For instance, we may require that two separate ports emit two given events in a fixed order. The two local testers, checking the occurrences of the events at the two ports, are by themselves unable to tell the order in which the events have been produced. In other words, the causality between the two events is not observed by the two testers. This kind of problems are known as controllability and observability problems. Being able to device test sequences that avoid them is of great interest. With such sequences, the testers can determine, for instance, when to apply its own inputs and whether an output observed is received in response to the correct input.

The paper "Implementation relations and test generation for systems with distributed interfaces" by Robert M. Hierons, Mercedes G. Merayo, and Manuel Nunez, is an in-depth study of the problems. Notions of conformance for an input-output transition system that has multiple ports are explored, adapting the widely used **ioco** implementation relation to this situation. An algorithm is introduced for checking if a test case is controllable and another one for generating a sound and complete test suite for the system under test.

An alternative approach is to allow the local testers, placed on the different ports, to exchange coordination messages among themselves. In this case, the test sequences should include such external coordination messages. Their addition, however, may have a cost, due to possible delays so introduced, and reducing these costs becomes a further challenge. The whole approach is explored in the paper "Overcoming Controllability Problems in Distributed Testing from an Input Output Transition System", by Robert M. Hierons. The author proposes a polynomial time algorithm for deciding controllability and a second polynomial time algorithm which avoids controllability problems by adding coordination messages. Moreover, the problem of optimizing coordination messages is shown to be NP-complete.

Program synthesis is methodology whose goal is the automatic construction of a program that satisfies a given specification. The paper "Symbolic Synthesis of Masking Fault-Tolerant Distributed Programs" by Borzoo Bonakdarpour, Sandeep S. Kulkarni, and Fuad Abujarad, uses program synthesis to generate masks for faulty distributed systems. Masking fault-tolerance means guaranteeing that programs continue to satisfy their specification even in the presence of faults.

When a complex software system is distributed and run, it may produce faults that had not been originally expected. It may also be that the requirements for the system have changed, so that certain patterns of behaviour, previously considered valid, are now regarded as faults. In these situations, stopping the system, modifying it, and reinstalling it, may be very costly. An alternative approach is to place masks, that is, additional components that hide or correct the undesired behaviours. Unfortunately, the problem of synthesizing masking fault-tolerant distributed programs is NP-complete in the size of the program's state space. The main contribution of the paper is showing that, in spite of the high worst-case complexity, the synthesis problem is feasible in practice, both concerning time and space. Precisely, the authors propose a BBD-based synthesis heuristic for automatically adding fault-tolerance masks. The effectiveness of the proposal is checked an various experimental cases studies. The limitations and strengths of the approach are then discussed. For instance, several bottlenecks of synthesis are identified, that depend on the structure of the program in input.

The above papers were regularly submitted, and went through the usual high-standard and stringent review procedure of Distributed Computing. I believe the readers of the journal, possibly working on other areas, will find them interesting.

It has been a great pleasure for me to see an issue of the journal dedicated to concurrency theory, and to write a preface for it. I would like to thank all those who contributed. First, thanks to Professor Hagit Attiya, the Editor-In-Chief of Distributed Computing, for suggesting an issue in concurrency theory. Then, my gratitude goes to the authors of the paper, for submitting their work to the journal, and to the referees who devoted time to enable a rigorous peer review.