State-Driven Testing of Distributed Systems: Appendix

Domenico Cotroneo, Roberto Natella, Stefano Russo, Fabio Scippacercola

Università degli Studi di Napoli Federico II
{cotroneo,roberto.natella,sterusso,fabio.scippacercola}@unina.it

Abstract. This appendix describes the system model that is adopted for the experimental evaluation of our state-driven testing approach. Moreover, it contains the glossary of the terms used in the paper.

1 System model

The system model adopted for state-driven testing is showed in Figure 1. Transitions in the Petri Net represent events that can occur during an execution. These events are logged by the Workload Generator during an execution, in order to infer the evolution of the system and to identify whether a target state has been reached. Events are logged by instrumenting the application code in key points, such the entry and exit points of methods of CORBA objects. Table 1 briefly describes the meaning of the considered events: they are related to requests issued by processes in the system, the completion of requests, and lock/unlock operations performed by the Façade on FDP Tables.

In Figure 1 transitions and places are grouped on the basis of their relationships with components of the FDPS (Client, Façade, Processing Server, Load-Balancing Service). The places in the uppermost part of the system model represent the state of the FDP Table in the Façade process: in particular, the number of tokens in places $A_1, \ldots, A_6$ represent the number of enqueued requests for the FDP number $1 \ldots 6$. When a request is sent from the Façade to a Processing Server, a token is removed from one of the places $B_1, \ldots, B_6$, and a token is added in the place $BF_1$. In turn, a token is added to one of the places $WRK_1, WRK_2, WRK_3$ according to a load-balancing strategy, which reflect the state of Processing Servers (busy or idle). When a Processing Server finishes, it invokes a callback method of the Façade, which unlocks the FDP and allows the system to process the next pending request for that FDP.
Fig. 1. System model of the FDPS.
Table 1. FDPS events considered in the system model.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client_FDP*</td>
<td>A client issued a request for an FDP</td>
</tr>
<tr>
<td>Lock_FDP*</td>
<td>The Façade locked an FDP; other requests cannot access the FDP until it is unlocked</td>
</tr>
<tr>
<td>PS_enqueue_FDP*</td>
<td>A request has been forwarded to the group of Processing Servers</td>
</tr>
<tr>
<td>PS_working_Host*</td>
<td>A Processing Server started processing a request</td>
</tr>
<tr>
<td>PS_end_working_Host*</td>
<td>A Processing Server finished processing a request</td>
</tr>
<tr>
<td>Façade_callback_FDP*</td>
<td>The Façade has been notified about the completion of a request</td>
</tr>
<tr>
<td>Unlock_FDP*</td>
<td>The Façade unlocked an FDP</td>
</tr>
<tr>
<td>Client_notify_FDP*</td>
<td>A client has been notified about the completion of a request</td>
</tr>
</tbody>
</table>

2 Glossary

In order to help the reader, this section summarizes the main terms used in the paper.

**Execution report** An execution report \( r \in R_W \) is the sequence of all the states traversed by the system under test in an execution under the workload \( W \), where each state traversal is denoted with \( (s_k, d_k) \) where \( s_k \in S \) is the state traversed and \( d_k > 0 \) is the sojourn time.

**Hard-to-reach state** A state is said hard-to-reach if it is not a trivial and easy-task to identify a workload that let the system to cross that state.

**Probability test success** It \( s \) is an estimation of the probability that the SUT, provided with the workload \( W \), meets the test triggering specification. It can be empirically estimated from the execution reports collected during the loops of the workload search phase.

**Process** The entity that communicates in a distributed system. The processes do computational steps and interact through a network. Each process has its own clock and encapsulates resources.

**Service** A service \( u_i \in U \) is a functionality that the system under test provides to its clients and it is offered by one interface. Each service can be invoked by the clients and it triggers different functionalities and actions in the DS depending on the actual values of the parameters sent in its requests.

**Service parameters** The service parameters \( m_{i,j} \in M_{u_i} \) are the combinations of actual values sent in the service requests for the service \( u_i \).
Service request A service request for the service \( u_i \) is a message produced by a client for the system under test with parameters \( m_{i,j} \) at time \( t \in T = \{0, \ldots, t_{\text{max}}\} \) of the experiment, and can be represented by a pair \( r \in M_{i} \times T \).

State The state of the process and of the DS is determined by the tester according to some high-level specification of the system, which takes into account the state of local resources as well as the state of computations performed by each process and the aim of the testing. An example of local state of a process could be down if the process is failed, or up otherwise; or initializing and waiting for ACK to distinguish between different states of a computation. The global state of the system is specified by the tester through a system model.

State-Driven Workload Generation The aim of SDWG consists in searching for a workload \( W \) such that the likelihood that the system under test spends at least a period \( \tau \) in the target state is high enough to allow the accurate and reproducible test execution in the desired state.

State-based testing State-based testing is a verification technique that aim to assess the system under several different conditions, by letting the system to cross some operational states and to evaluate the system behavior in that states.

System model The system model is a high-level specification (using a formalism such as Finite State Machines (FSM) or Petri Nets (PN)) by which the tester describes the set of global states, including the target states. The tester should define the system model on the basis of system requirements and of its high-level design (such as, state and interaction diagrams, and tester’s knowledge about protocol specifications), and should take into account the state of local resources and the state of computations at each process, as well as the goals of testing.

Target state A system state object of interest during a state-based test. The definition of target state has to be provided according to the system model. For instance, in case that the system model is a Petri net, the target state can be formulated in terms of constraints that the marking must fulfill.

Test a system state The action of performing a state-based test in a specified target state.

Test executor It is a Tester’s process or module waiting for a test trigger during a state-based experiment in order to perform an action or an observation when the SUT is in the target state.

Test trigger It is the event raised by the Workload Generator Framework during an experiment for the Test Executor module when the test triggering specification is satisfied by the system under test. When a test trigger is thrown, the system must sojourn in the target state for at least the time \( \tau \) specified by the Tester in the SDWG phase. The test trigger is raised only if the triggering delay is over.

Test triggering specification It is a specification of the conditions that must hold during a testing session in order to raise a test trigger. For instance, it can be the description of the target state and of the triggering delay.
**Testing-phase** It is one of the two phases of our approach for the State-Driven testing. In this phase the Workload Generation Framework provides the system with the workload $W$ and raises test triggers when the test trigger specification is met during the execution. At the end of an execution the WG collects the system logs and performs a post-mortem analysis of the traces in order ascertain the correct performing of the experiment.

**Triggering delay** It is a timer used for filtering out test triggers when, during an experiment, the test triggering specification does not hold more than the triggering delay timeout. The Workload Generator Framework set in the workload search-phase the triggering delay in order to maximize in the testing-phase the probability of test success.

**Workload** A *workload* is a set of service requests generated during a system execution, and it is a subset $W$ of the set $W^*$ representing the space of all possible requests that can be submitted to the system: $W \subseteq W^* = \bigcup_{u \in U} \{ (m_{i,j}, t), m_{i,j} \in M_u, t \in T \}$. A workload is an element of the powerset (the set of all subsets) of $W^*$, that is, $W \in \mathcal{P}(W^*)$.

**Workload Generator Framework** It is the system developed to implement our approach for the state driven testing. It realizes both the workload search phase and the test phase.

**Workload Search-phase** It is one of the two phases of our approach for the State-Driven testing. In the Workload Search-phase the Workload Generator Framework performs a state-driven workload generation: it interacts with the DS under test in a closed-loop configuration, it first exercises the DS with a workload, then analyzes its behavior, and modifies the workload until a specified target state is reached. In this loop, the WG alternates an on-line phase, in which the DS is executed, and an off-line phase, in which the behavior of the DS is analyzed. When the experiment is over, an off-line synchronization algorithm is executed to align the events of an execution on a single global timeline.