State Machine Based Simulation Testing

Florian Lier[1], Norman Köster[1], Ingo Lütkebohle[2], and Sven Wachsmuth[1]

[1] Center of Excellence Cognitive Interaction Technology (CITEC), Universitätsstraße 21-23, 33615 Bielefeld, Germany
[2] CoR-Lab Research Institute for Cognition and Robotics, Universitätsstraße 25, 33615 Bielefeld, Germany

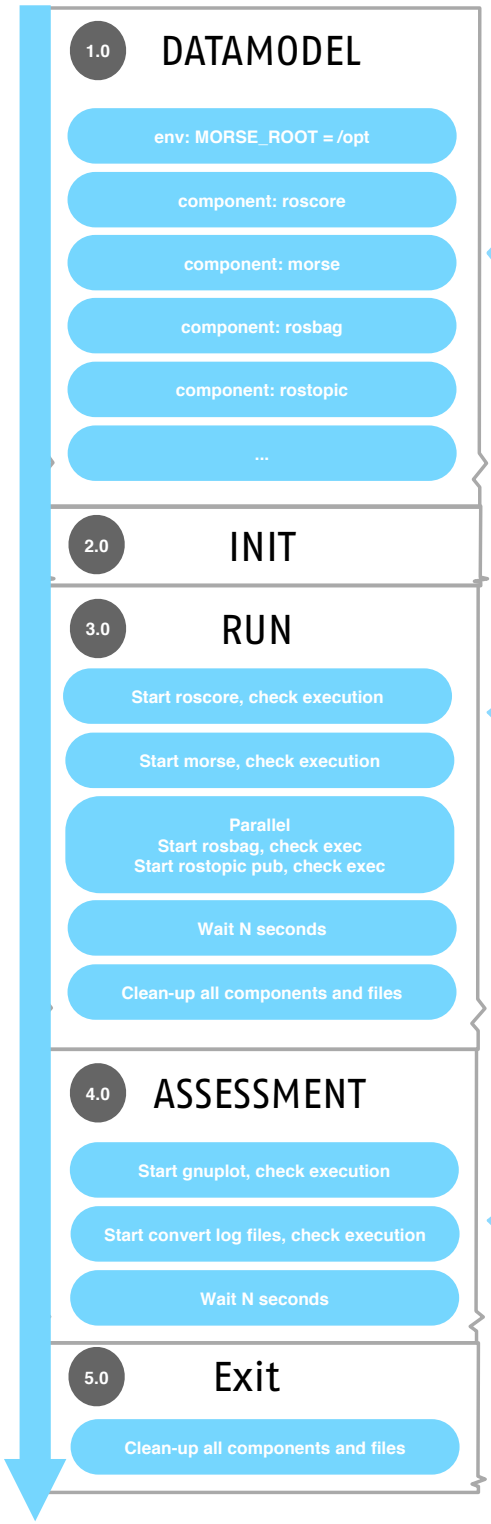
Robot simulators, like the MORSE[1] project, provide a safe and readily available environment for robot system testing, reducing the effort for testing drastically. In principle, simulation testing is automatable, and thus a good target for Continuous Integration (CI)[2] testing. However, so far, high-level scenario tests still require complex component setup and configuration[3][4] before they can be run in the simulator. An added complication is, that there is no standard for starting, configuring, or monitoring software components on todays robots. Often, high-level tests are carried out manually, implementing a tailored solution, e.g, via shell scripts or launch files[5], for a specific system setup. Besides the effort of manual execution and supervision, current tests mostly do not take timing and orchestration, i.e., required process start-up sequence, into account. Furthermore, successful execution of components is not verified, which might lead to subsequent errors during the execution chain. Most importantly, all this knowledge about the test and its environment is implicit, often hidden in the actual implementation of the tailored test suite.

To overcome these issues, this contribution introduces a generic and configurable state-machine based process to automate a) *environment setup*, b) *system bootstrapping*, c) *system tests*, d) *result assessment*, and e) *exit and clean-up strategy*. We have chosen a state-based approach in order to inherit a well structured automaton, which enables us to invoke the steps mentioned above, in the desired order, and to explicitly model required test steps. Furthermore, the state-chart model[6] enables us to invoke states in parallel, or sequentially, which also makes orchestration, e.g., start-up of system components, feasible and most importantly — controllable. Last but not least, errors during the execution will prematurely end the state-machine to prevent subsequent errors.

In the following we will provide an exemplary use-case: driving an ATRV robot using MORSE as a simulation environment[7], and ROS for middleware purposes. Figure 1 depicts the five-layered structure of our state-chart-based test representation. These five layers consist of: 1) a datamodel section, 2) initialization state, 3) run state, 4) assessment state, and 5) exit and cleanup state, which are executed sequentially. Firstly, all software components and their environment configuration, required for the simulation test, are defined in the "datamodel" section. Besides others, we define the environment variable *MORSE_ROOT* in the "environment" block, and a *roscore* component in the "software" section for instance. The actual system start-up is accomplished in the run state, by executing previously defined software components — sequentially and/or in parallel, within their defined environment. In order to ensure successful execution of each

component, observer processes are spawned, ascertaining individual local criteria, such as the existence of the *PID*, or defined prints to *STDOUT*. In case of a single unsatisfied criteria, e.g., *PID* not existent, the state-machine is configured to transition into an error state, and the test is aborted to prohibit subsequent errors. Once the system start-up is finished successfully, the state-machine transitions into a wait state. In the underlying ATRV use-case this state lasts for thirty seconds, in which multiple movement commands are issued via ros topics. In parallel, a rosbag and several ATRV motion related topics are saved to log and rosbag files. After the wait phase, a transition to the assessment state follows. In order to evaluate previously recorded results, a clean up is done initially, properly ending all previously started components, e.g., morse, roscore, logging tools, etc. Similar to the run state, software components, required for the analysis, e.g., plotting tools can be launched in the result assessment state as well. Lastly, the state-machine transitions to the exit phase, consisting of a final clean up including ending of all remaining software components, file handlers, and so on.

This configuration can be easily run on a CI server, as all files and components are started, orchestrated and closed by the state-machine automatically. The results can be saved as so-called "build artifacts", e.g., plots, videos, images, or even logged results converted into xUnit[8] compatible output. Besides automated CI tests, state-machines might be launched in the cloud[9] and developers may interact, in real-time, with the simulation for a previously defined time, e.g., through an extensive wait state. Last, but not least, developers might setup a repository of scxml test files to share and provide examples, and to compare results, based on common, well-defined test definitions and executions.




```
<datamodel>
    <data id="environment" xmlns="http://my_fsm.com">
        <variable var="PREFIX" val="/vol/robocup/opt/"/>
        <variable var="MORSE_ROOT" val="/vol/robocup/opt/"/>
        <!-- Further variables ... -->
    </data>
    <data id="software">
        <component val="roscore">
            <command val="roscore"/>
            <path val="/opt/ros/groovy/bin/"/>
            <executionHost val="localhost"/>
            <checkExecution val="True">
                <checkType val="pid" timeout="8"
                                     blocking="True"
                                     ongoing="True"/>
                <checkType val="stdout" timeout="8"
                                      criteria="started core service"
                                       blocking="True"/>
            </checkExecution>
        </component>
        <component id="morse"><!-- Morse content ...--></component>
        <!-- Further components ...-->
    </data>
</datamodel>
```

```
<state id="run_test" initial="ROS_startup">
    <!-- Allows to catch errors in sub-states -->
    <transition event="unsatisfiedCriteria" target="criteriaError" >
        <log label="ERROR" expr="'recieved unsatisfiedCriteria evt'"/>
    </transition>
    <state id="ROS_startup">
        <onentry>
            <log label="INFO" expr="'Entering State ROS Startup'" />
            <my_fsm:executeProgram value="roscore" />
        </onentry>
        <transition event="executeProgram.success" target="morse"/>
    </state>
    <!-- Further components to be executed ... -->
    <state id="recordangles">
        <!-- ... --->
        <transition event="executeProgram.success" target="wait"/>
    </state>
</state>
```

```
<state id="result_assessment">
    <initial>
        <transition target="clean"/>
    </initial>
    <transition event="unsatisfiedCriteria" target=".." />
    <state id="clean">
        <onentry>
           <my_fsm:cleanUp expr="my_executable" value="" />
        </onentry>
        <transition target="compute_results"/>
    </state>
    <!-- Assessment components are started here ... -->
    <state id="waitfinish">
       <onentry>
           <send event="wait.finish" delay="'30s'" />
       </onentry>
       <transition event="wait.finish" target="exitTest"/>
    </state>
</state>
```

Fig. 1: Schematic state-machine based system test.

1 Acknowledgments

This work has been partially supported by the German Aerospace Center (DLR) with funds from the Federal Ministry of Economics and Technology (BMBF) due to resolution 50RA1023 of the German Bundestag and by the German Research Foundation (DFG) within the excellence program EC 277 (Cognitive Interaction Technology – CITEC).

References

1. Gilberto Echeverria, Sverin Lemaignan, Arnaud Degroote, Simon Lacroix, Michael Karg, Pierrick Koch, Charles Lesire, and Serge Stinckwich. Simulating complex robotic scenarios with morse. In *SIMPAR*, pages 197–208, 2012.
2. Martin Fowler and Matthew Foemmel. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, 2006.
3. D. Brugali and P. Scandurra. Component-based robotic engineering (part i) [tutorial]. *Robotics Automation Magazine, IEEE*, 16(4):84 –96, december 2009.
4. D. Brugali and A. Shakhimardanov. Component-based robotic engineering (part ii). *Robotics Automation Magazine, IEEE*, 17(1):100 –112, march 2010.
5. Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 2009.
6. Jim Barnett, Rahul Akolkar, RJ Auburn, Michael Bodell, Daniel C Burnett, Jerry Carter, Scott McGlashan, Torbjörn Lager, Mark Helbing, Rafah Hosn, et al. State chart xml (scxml): State machine notation for control abstraction. *W3C Working Draft*, 2007.
7. LAAS-CNRS. Ros and morse tutorial. *http://goo.gl/ro2Ao*, 2013.
8. Gerard Meszaros. *xUnit test patterns: Refactoring test code*. Addison-Wesley Professional, 2007.
9. Open Source Robotics Foundation. Cloudsim. *http://gazebosim.org/wiki/CloudSim*, 2013.