CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

REVERT: A Monitor Generation Tool for Real-Time Systems

Sangeeth Kochanthara
Geoffrey Nelissen
David Pereira
Rahul Purandare

CISTER-TR-161007

REVERT: A Monitor Generation Tool for Real-Time Systems

Sangeeth Kochanthara, Geoffrey Nelissen, David Pereira, Rahul Purandare

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

Abstract

We present REVERT (which stands for RuntimE VErification for Real-Time systems), a new tool to generate monitorsfor real-time systems. REVERT takes specifications written ina new Domain Specific Language (DSL) and automaticallygenerates monitors under the form of complete timed deterministic finite automata (DFA). The generated timed DFA canlater be used to generate code that can eventually be integratedwithin the monitored system.

REVERT: A Monitor Generation Tool for Real-Time Systems

Sangeeth Kochanthara
IIIT-Delhi,
India

Geoffrey Nelissen
CISTER/INESC TEC, ISEP,
Portugal

David Pereira
CISTER/INESC TEC, ISEP,
Portugal

Rahul Purandare
IIIT-Delhi,
India

I. INTRODUCTION

We present REVERT (which stands for RuntimE VErification for Real-Time systems), a new tool to generate monitors for real-time systems. REVERT takes specifications written in a new Domain Specific Language (DSL) and automatically generates monitors under the form of complete timed deterministic finite automata (DFA). The generated timed DFA can later be used to generate code that can eventually be integrated within the monitored system.

II. REVERT LANGUAGE

The REVERT specification language is specially crafted for RTS with an implicit notion of time. The functional as well as non-functional properties are written using a combination of state machines, extended regular expressions (ERE), boolean expressions, and high-level timing operators (`time` and `duration`). Functional properties like events execution orders are specified using ERE, while non-functional properties like, timing constraints on a sequence of events and execution time of jobs are specified using the `time` and `duration` operators, respectively. Each job is defined using the following four sets of events: `start`, `suspend`, `resume` and `complete`, which contain events related to the release, suspension (for instance, due to preemption, unavailability of a shared resource or a self-suspension), resumption, and completion of the job, respectively.

Listing 1 shows the outline structure of a monitor specification in REVERT. The `observe` statement specifies the events that are monitored by the monitor m_i. The events listed here are a subset of all the events monitored in the system. The `jobs` statement declares jobs using the four sets of events discussed above. The `nodes` statement declares different states of the monitor. These nodes can be seen as different specifications that must be monitored in different modes of execution of the system, for example, taxi, takeoff, cruise, and landing modes of a plane. The node in which the monitor starts its execution is declared using the `initial` statement. The nodes are later defined in `node` blocks with `trans` statements defining the transitions between nodes, guarded by monitored properties. These properties are written inside the `prop` block using ERE and boolean expressions on the operators `time` and `duration`. Computational code reacting to some specific properties can be added to the specification. The `init` block declares code executed when transitioning to a node, while `trans` provides means to declare code when transitioning out of the node.

```
monitor m_i {
  observe { ev_1, ..., ev_l }
  jobs{
    j_1{
      start: {ev_1, ev_2}
      suspend: {ev_3}
      resume: {ev_1}
      complete: {ev_6}
    },
    ...,
    j_p{...}
  }
  nodes { n_1, ..., n_k }
  initial { n_q }

  node n_1 { init_1 prop_1 trans_1 }
  ...
  node n_k { init_k prop_k trans_k }
}
```

Listing 1: Structure of a monitor specification in REVERT

III. MONITOR GENERATION

The tool transforms specifications written in the proposed DSL to a complete timed automaton. This automaton can then be used to check if traces monitored during the system execution respect the given specifications. The monitor automaton generated by the presented tool is saved in an xml format. A graphical representation of the monitor automaton is also generated.

IV. DEMONSTRATION

The tool will be showcased through the following steps.

1) How to craft a specification using the REVERT DSL followed by explanation of an example specification.
2) Conversion of the example specification to a monitor automaton using the tool.
3) Correlation between the xml and the graphical representation of the monitor automaton.
4) Demonstration of changes in the monitor automaton according to changes in the specification.

V. ACKNOWLEDGMENTS

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within the CISTER Research Unit (CEC/04234); also by ARTEMIS/0001/2013 - JU grant nr. 621429 (EMC2).

```
use"T_Events.ev";
use "Ext_Procs.h";

monitor MyMon {
      observe { startT,suspT, blockedT, resumeT, unblockedT,complT,arrT}

       jobs {
             Job2 {
                  start: {startT}
                  suspend: {suspT, blockedT}
                  resume: {resumeT, unblockedT}
                  complete: {complT}
             }
       }

       nodes { Normal, Recovery}
       initial { Normal }

       node Normal {
             init{
                  resetAllSystemFlags();
             }
             constraints
              {
                     c2: duration(Job2) < 10;
                     c5[ERE]:  arrT * blockedT  ;
              }
              transitions {
                        fail_duration: failure(c2) -> Recovery{
                              triggerJob2Correction();
                         }
                        fail_ERE: failure(c5) -> Recovery{
                              setErrorFlags();
                        }
             }
      }

       node Recovery {
              init{
                        initializeSystemRecovery();
              }
              constraints {
                     c1[ERE]:  arrT+ _ blockedT  ;
                     c31:time(arrT blockedT*) <= 534 ;
              }
              transitions {
                     success_ERE: success(c1) -> Normal;
                     success_time: success(c31) -> Normal{
                           resetErrorFlags();
                        }
              }
      }
}
```

Fig. 1: Example REVERT specification

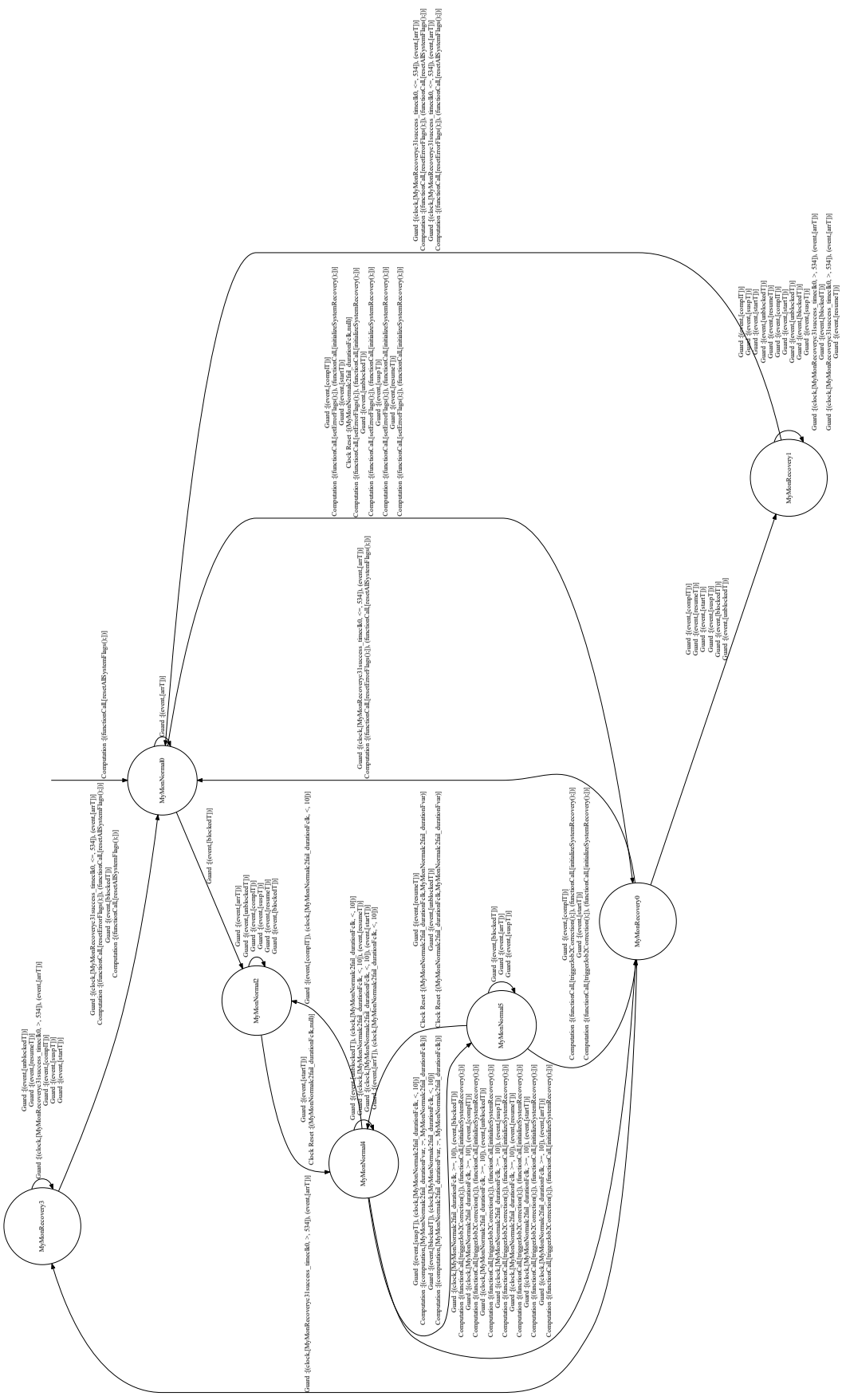

Fig. 2: Graphical representation of the monitor generated from example specification in Fig. 1

```
<?xml version="1.0" encoding="UTF-8"?>
<automaton name="MyMon">
    <event_files list_of_files="T_Events.ev"/>
        <!-- Set of states of the monitor -->
  - <states>
        <state name="MyMonRecovery3"/>
        <state name="MyMonRecovery1"/>
        <state name="MyMonRecovery0"/>
        <state name="MyMonNormal4"/>
        <state name="MyMonNormal5"/>
        <state name="MyMonNormal0"/>
        <state name="MyMonNormal2"/>
    </states>
        <!-- Set of events used in the monitor -->
  - <events>
        <event name="complT"/>
        <event name="unblockedT"/>
        <event name="blockedT"/>
        <event name="resumeT"/>
        <event name="suspT"/>
        <event name="arrT"/>
        <event name="startT"/>
    </events>
        <!-- Set of clocks used by the monitor -->
  - <clocks>
        <clock name="MyMonNormalc2fail_durationFclk"/>
        <clock name="MyMonRecoveryc31success_timeclk0"/>
    </clocks>
        <!-- Set of initial computations -->
  - <initial_computations>
        <function from="Ext_Procs.h" call="resetAllSystemFlags();"/>
    </initial_computations>
        <!-- Transitions -->
  - <transitions>
    - <transition dst="MyMonNormal0" src="MyMonRecovery3">
      - <guard>
            <event name="arrT"/>
        - <clocks>
          - <clock name="MyMonRecoveryc31success_timeclk
                <condition_type>"leq"</condition_type>
                <value>"534"</value>
            </clock>
        </clocks>
      </guard>
      - <actions>
        - <computations>
            <function from="Ext_Procs.h" call="resetErrorFlag
            <function from="Ext_Procs.h" call="resetAllSystem
        </computations>
      </actions>
    </transition>
    - <transition dst="MyMonRecovery0" src="MyMonNormal0">
      - <guard>
            <event name="complT"/>
        </guard>
      - <actions>
        - <computations>
            <function from="Ext_Procs.h" call="setErrorFlags(
            <function from="Ext_Procs.h" call="initializeSysten
        </computations>
      </actions>
    </transition>
    - <transition dst="MyMonNormal0" src="MyMonRecovery1">
      - <guard>
            <event name="arrT"/>
        - <clocks>
          - <clock name="MyMonRecoveryc31success_timeclk
                <condition_type>"leq"</condition_type>
                <value>"534"</value>
            </clock>
        </clocks>
      </guard>
      - <actions>
        - <computations>
          - <clock name="MyMonNormalc2fail_durationFclk">
                <value>"0"</value>
            </clock>
        </clocks>
      </actions>
    </transition>
  - <transition dst="MyMonRecovery0" src="MyMonNormal4">
    - <guard>
        <event name="resumeT"/>
      - <clocks>
        - <clock name="MyMonNormalc2fail_durationFclk">
            <condition_type>"geq"</condition_type>
            <value>"10"</value>
          </clock>
        </clocks>
      </guard>
    - <actions>
      - <computations>
          <function from="Ext_Procs.h" call="triggerJob2Correction();"/>
          <function from="Ext_Procs.h" call="initializeSystemRecovery();"/>
        </computations>
      </actions>
    </transition>
  - <transition dst="MyMonRecovery0" src="MyMonNormal4">
    - <guard>
        <event name="startT"/>
      - <clocks>
        - <clock name="MyMonNormalc2fail_durationFclk">
            <condition_type>"geq"</condition_type>
            <value>"10"</value>
          </clock>
        </clocks>
      </guard>
    - <actions>
      - <computations>
          <function from="Ext_Procs.h" call="triggerJob2Correction();"/>
          <function from="Ext_Procs.h" call="initializeSystemRecovery();"/>
        </computations>
      </actions>
    </transition>
  - <transition dst="MyMonRecovery0" src="MyMonNormal4">
    - <guard>
        <event name="arrT"/>
      - <clocks>
        - <clock name="MyMonNormalc2fail_durationFclk">
            <condition_type>"geq"</condition_type>
            <value>"10"</value>
          </clock>
        </clocks>
      </guard>
    - <actions>
      - <computations>
          <function from="Ext_Procs.h" call="triggerJob2Correction();"/>
          <function from="Ext_Procs.h" call="initializeSystemRecovery();"/>
        </computations>
      </actions>
    </transition>
  - <transition dst="MyMonNormal2" src="MyMonNormal2">
    - <guard>
        <event name="blockedT"/>
      </guard>
    </transition>
  </transitions>
        <!-- Initial state -->
  <initial_state name="MyMonNormal0"/>
        <!-- Final states -->
  <final_states> </final_states>
</automaton>
```

Fig. 3: Start and end of xml representation of the monitor generated from the specification in Fig. 1