Application of Early Error Detection for Handling Degraded Modes of Operation

Thomas Robert, Jean-Charles Fabre, Matthieu Roy

Application of Early Error Detection for Handling Degraded Modes of Operation

Thomas Robert[1,2], Jean-Charles Fabre[1,2], Matthieu Roy[1,2]

[1]LAAS-CNRS; Université de Toulouse
7 avenue du colonel Roche,
F-31077 Toulouse, France
[2]Université de Toulouse; UPS, INSA, INP, ISAE ; LAAS ;
F-31077 Toulouse, France
{trobert, fabre, mroy}@laas.fr

Keywords: timed automata, monitoring for early detection, fault tolerance design pattern.

1. Introduction

Real-time software applications are in charge of maintaining a timely and predictable interaction with their environment, being it hardware devices, or other software entities. In this paper, we illustrate the use of early error detectors (that we previously described in [1]), generated from timed automata based specifications, in a recovery block-like design pattern. This design pattern is enhanced with our Early Error Detection service that allows triggering as soon as possible error recovery mechanisms*.

2. The design pattern

We propose an extension of classical design pattern for error detection and recovery in the context of real time systems. The recovery block design pattern, [2], consists of one fault-tolerant component that can carry out a real time service. The design pattern is defined as follows:

- The service is implemented in a recovery block containing several sub-blocks called alternates in the seminal paper. Each one delivers a service considered as satisfactory. Each block execution is time bounded.
- A block is executed as long as no error is detected. When an error is detected in a block, it triggers the execution of a different sub-block.
- In addition to internal error detection services nested in each block, the recovery block contains an acceptance test and alarms. Alarms are used to ensure bounds on block execution time. The acceptance test is the last barrier to prevent error propagation. It checks if the blocks outputs are valid. This test is a sort of oracle that each block should be able to pass.

In practice, the sub-blocks are degraded operational modes corresponding to simpler activity with a degradation of the service quality. This degradation is accepted as such degraded operational modes are easier to schedule as they can be executed faster. The faults that caused the errors already observed in a mode should not be activated in any other degraded mode. Thus, the acceptance test may be difficult to implement as block behaviors may be very different.

We propose to extend this conventional design pattern with behavioral models attached to each block. These models will be used to generate detectors that raise error signals as soon as the behavior of blocks do not correspond to their specification, i.e., implementing Early Error Detection (EED). Then block execution is interrupted and the next block is activated. The figure below represents the structure of such a fault tolerant component.

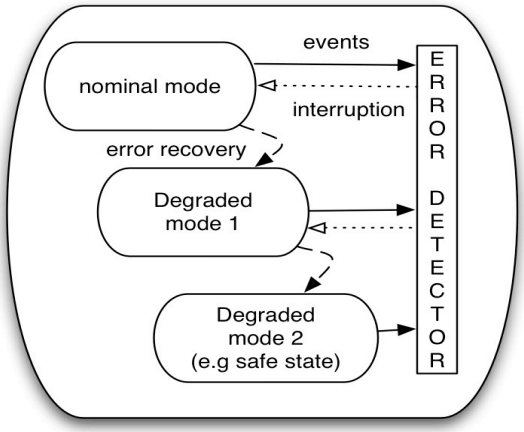

Figure 1 Design pattern for execution of degraded operational modes

As said in [2], the efficiency of such a design pattern highly depends on the quality of the error detection services. We will first provide a motivating example of the application of Early Error Detection and then explain how it is integrated in the design pattern.

3. Specification of real-time behaviors

The allowed behavior of each block is described via a timed automaton [3]. Such automaton defines allowed executions as a set of traces. These traces describe the sequences of allowed events with their timing constraints.

* This work has been partially supported by RESIST, Resilience in IST, the Network of Excellence n°026764

Liveness constraints are defined as conditions on the final states in which the block is allowed to stop in the automaton. A trace is *valid* if and only if a final state can be reached when this trace is executed on the automaton. An example of such timed automaton is provided in Figure 2. By definition, the specification of this component is the set of timed traces that are *recognized* by this automaton.

Intuitively, such an automaton has its transitions labeled with *events* (e.g., ***release***, ***end***, etc.) and timing constraints. Timing aspects are handled using *clocks*. The above example uses two clocks, *x* and *y*. Each transition has three labels: an event, a guard, that is a logic formula on *clocks* (e.g., *x<20*), and a set of *clocks* to be reset when the transition is fired. A complete description of this formalism can be found in [3].

4. Example specification

The automaton illustrated in Figure 2 corresponds to an application that periodically checks if the position of a flap should be changed given a value measured on a sensor.

Double circles represent final states, and the initial state is marked with a disconnected arrow. Here, we consider a periodic task that is constrained by a global deadline of *20 ms*. The event ***release*** identifies the triggering of the task within a period. It can only be accepted when the task is already finished.

In the first step, the application reads a value on a sensor. By design, this operation has been made time predictable at the expense of locking the sensor for at least *10 ms*. Thus, the step 1 cannot be re-executed for the next *10 ms*. The second step consists of analyzing the sensed value to decide whether the flap state should be changed. In these two steps, the event ***chg*** identifies an important change in the environment of the sensor, and triggers the re-execution of these two steps.

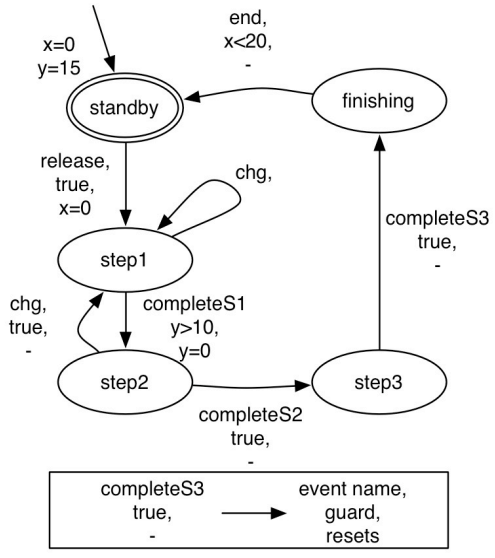

Figure 2 Timed automaton based specification

The third step corresponds to the generation, if required, of the state change of the flap.

Each step change is identified by an ending event, namely ***CompleteS1***, ***CompleteS2***, and ***CompleteS3***. A last event, ***end***, identifies the end of the processing.

5. Early Error Detection

In [1] we showed how to automatically derive from such an automaton a detector that provides an *early detection service.*

Definition*: An error detector provides an early error detection service for a specification S if and only if it produces an error event* EED_S *as soon as the observed trace cannot be continued in a valid one, i.e., a final state can no longer be reached, whatever happens next.*

It is worth noticing that traditional simple greedy approaches that fire transitions when events occur cannot provide early error detection service, i.e., are unable to detect an error as soon as possible.

Such a service is implemented following the observer worker architecture, [4], the detector being the observer, and the block being the worker.

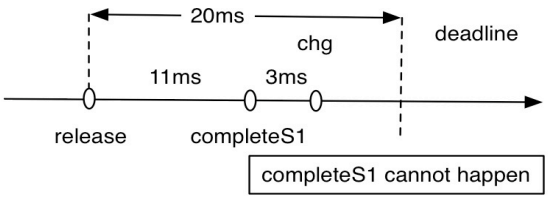

Figure 3 Illustration of Early Error Detection

To show why traditional algorithms cannot provide early detection of errors, let us consider the execution scenario depicted in Figure 3. Suppose that Step 1 has been delayed and is completed *11 ms* after the ***release*** event. By the specification, the sensor cannot be reused for the next *10 ms*. Now if a ***chg*** event is received, there is no way to accommodate at the same time the deadline and the minimal locking time of the sensor: Step 1 should be completed within *20* time units from the ***release*** event whereas the sensor cannot be reused before *21* time units from the same ***release*** event. Thus, an error should be signaled. This example shows why the timed automaton cannot be used as is in the observer pattern, i.e., a basic lazy approach cannot detect errors early. Notice that on the timed automaton the transition ***chg*** is never explicitly disabled. Such a situation often happens in complex models where the specification aims at minimizing redundant constraints.

Our Early Error Detection framework pre-processes the automaton off-line so that the observer used on-line correctly triggers an error as soon as possible, and in the execution scenario depicted above, just when Step 1 finishes after *11ms*.

6. Interest for handling degraded modes of operation

To integrate Early Error Detection in the design pattern described before, we use the special event EED_S to trigger the "recovery transition" from one block to another, i.e., from one operational mode to another.

Following the example previously given, we show in Figure 4 that the nominal mode of operation is triggered first. According to the scenario explained in Figure 3, the $EED_{nominal}$ event is obtained at time *11ms*. The event triggers the execution of block B1 that is a weak implementation of the flap control function. Several degraded modes of operation can be defined, corresponding to weaker and weaker implementations with stronger timing constraints. The nesting of such degraded modes of operation is possible when the remaining time allows it.

Thanks to the integration of the Early Error Detection approach in the design pattern, the remaining time for recovery is optimal with respect to error situations.

The last block, here block B2 can thus be a stop of the system in a pre-defined safety state, say placing the flap in medium position in our example.

Finally, the implementation of the proposed approach implies that all blocks in the pattern have a timed automaton describing their expected behavior.

7. Concluding remarks

The main benefit of this approach is that tentative error recovery can be triggered as early as possible. The EED approach has been integrated into the *Xenomai Real-Time Programming Interface*. In our current experiments, the cost of early error detectors is reasonable, not in terms of memory footprint, but in time overhead.

In the current state of the work, each block has independent specification and thus timing constraints. It would be of high interest to use hierarchical timed automata, [5], modeling the whole design pattern behavior. This means that the whole set of possible blocks is part of a single model in which global timing constraints can be defined. In this model, one can easily identify the modeling of each block, the transition between each block models being the *EED* special event. The benefit here is that the block to be activated next can be selected according to the remaining time, something that was not possible in the independent block modeling approach.

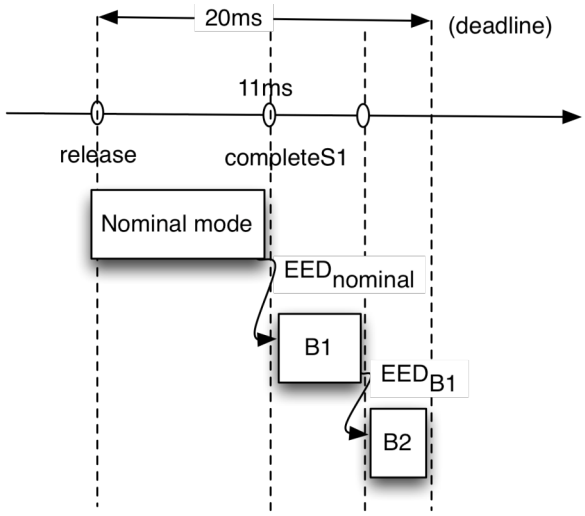

Figure 4 Triggering degraded modes of operation

8. References

[1] Thomas Robert, Jean-Charles Fabre, Matthieu Roy, "On-line Monitoring of Real Time Applications for Early Error Detection", Pacific Rim International Symposium on Dependable Computing, 2008, IEEE, pp. 24-31.

[2] B. Randel and J. Xu, "The Evolution of the Recovery Block Concept," Software Fault Tolerance, M.R. Lyu, ed., John Wiley & Sons, New York, 1995, chapter 1

[3] Alur, R. and D.L. Dill, "A theory of timed automata". Theoretical Computer Science, 1994. **126**: p. 183--235.

[4] M. Diaz, G. Juanole and J-P. Courtiat, "Observer-A Concept for Formal On-Line Validation of Distributed Systems", IEEE Transactions on Software Engineering, Volume 20 , Issue 12 (Dec' 1994)

[5] Jin, X., Ma, H., and Gu, Z. "Real-Time Component Composition Using Hierarchical Timed Automata". 7th international Conference on Quality Software, 2007, p. 1-10.