Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 133 (2005) 81–100
www.elsevier.com/locate/entcs

Modeling Fault-tolerant Distributed Systems for Discrete Controller Synthesis

Alain Girault, Eric Rutten[1]

INRIA Rhône-Alpes, POP ART,
655 avenue de l'Europe, 38334 Saint-Ismier cedex, FRANCE.

Abstract

Embedded systems require safe design methods based on formal methods, as well as safe execution based on fault-tolerance techniques. We propose a safe design method for safe execution systems: it uses discrete controller synthesis (DCS) to generate a correct reconfiguring system. The properties enforced concern consistent execution, functionality fulfillment (whatever the faults, under some failure hypothesis), and several optimizations. We propose model patterns for a set of periodic tasks, a set of distributed, heterogeneous and fail-silent processors, and an environment model that expresses the potential fault patterns. We outline an implementation of our method, using the Sigali symbolic DCS tool and Mode Automata.

Keywords: Discrete controller synthesis, fault-tolerance, real-time systems.

1 Introduction

1.1 Safety critical embedded systems

Embedded systems account for a major part of critical applications (space, aeronautics, nuclear...) as well as public domain applications (automotive, consumer electronics...). Their main features are:

- *duality automatic-control/discrete-event*: they include control laws modeled as differential equations in sampled time, computed iteratively, and discrete event systems to sequence the control laws according to mode switches;

[1] Email: Alain.Girault@inrialpes.fr, Eric.Rutten@inrialpes.fr

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2004.08.059

- *critical real-time*: unmet timing constraints may involve a system failure leading to a disaster;
- *limited resources*: they rely on limited computing power and memory because of weight and encumbrance, power consumption (autonomous vehicles or portable devices), radiation resistance (nuclear or space), or price constraints (consumer electronics);
- *distributed and heterogeneous architecture*: they are often distributed to provide enough computing power and to keep computing sites close to the sensors and actuators.

1.2 Problem statement

An embedded system being intrinsically critical, it is essential to insure that it is tolerant to processor failures. This can even motivate its distribution itself. In such a case, at the very least, the loss of one computing site must not lead to the loss of the whole application. We are interested in formal methods to model systems with guarantees on their fault-tolerance. Among the various existing formal methods, we investigate the use of *discrete controller synthesis* (DCS). The advantages of using DCS are the correctness of the resulting system and the easy modifiability of the controller (thanks to automatic tools), i.e., the possibility to study and test *several* fault-tolerance objectives or failure hypotheses on the same system model, without the need to re-design the system. Specifically, our objective is:

> *To produce automatically a controller enforcing fault-tolerance for a given distributed system.*

Fault-tolerance is the faculty to *maintain functionality of a system, whatever the faults* under some failure hypothesis. To achieve this, we will need first to model our distributed systems, and second to express formally some fault-tolerance objective, in terms of events and states of the system.

We propose to designers a methodology for modeling a system and studying the existence of fault-tolerant solutions according to several failure hypotheses and system's configurations. When a solution is found, it can be used either as a guideline for implementation (if the model was an abstract one [9]) or for deployment with a dynamic failure reconfiguring feature (this paper).

In our approach, a system consists of a set of tasks placed in a *configuration* onto a set of processors. Upon occurrence of a fault, one or several processors become unusable, and tasks must be placed anew in another configuration, by restarting them onto another processor, so that execution can proceed. These *reconfigurations* of the system have to be controlled according to a fault-tolerance policy, enforced by a *task manager*. The latter is specified in terms