CHESS CENTER FOR HYBRID & EMBEDDED SOFTWARE SYSTEMS

COLLEGE OF ENGINEERING, UC BERKELEY

Error Handling in Model-Based design for Real-Time Systems

Shanna-Shaye Forbes
Edward A. Lee

http://chess.eecs.berkeley.edu/

Work In Progress

Introduction

Designing effective error handling in an embedded software systems is essential for acceptable and reliable functionality in cases of errors and for the recovery from faults.

Errors in the error handling system can cause catastrophic failures of the software and can endanger human life. We take a principled approach of extending a model of computation (MOC) with timing semantics for embedded systems by an error handling mechanism for timing errors in model-based design.

Background

It is extremely important that one identifies and is capable of handling error cases before deploying embedded software.

One ineffective solution that has been used in the past is to have the system reset itself for every error encountered. If a system resets itself too often this can lead to significant loss of productivity, possible loss of data, and in extreme cases, possible loss of life.

Examples of the importance of error handling:

In 1996 the European Space Agency's Ariane 5 rocket self-destructed 40 seconds after launch. The underlying cause of the self-destruct sequence was a 64-bit floating point to 16-bit integer conversion exception. This occurred because of reuse of code designed for the much smaller Ariane IV [1] .

Image Source: http:// www.josefdobler.de /Neuer%20Ordner/ launch-site/ aml33h.jpg

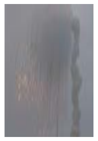


Image Source: http:// sspg1.bnsc.rl.ac.uk/SEG/ Ariane%205%20Explosion %20Report.htm

More recently, the SPIRIT Mars rover encountered a "reboot loop" shortly after landing, where a fault during the booting process caused the system to reboot again Luckily, a software patch solved the problem and the mission continued successfully[2].

Image Source: http://en.wikipedia.org/wiki/ File:NASA_Mars_Rover.jpg

A Royal Airforce pilot accidentally dropped a practice bomb on the flight deck of the Royal Navy's aircraft carrier. It missed it's intended target and several sailors were injured. The cause was attributed to a timing delay in the software.

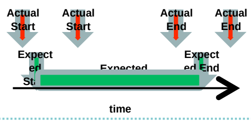
Image Source: http://upload.wikimedia.org/wikipedia/commons/d/dc/ Typhoon_f2_zj910_arp.jpg

In an effort to avoid possible similar mistakes with newly designed systems and to provide a more systematic means of dealing with timing errors, we present preliminary work that extends a model of computation (MOC) for embedded systems which features timing semantics.

Methodology

The focus of this work is on timing errors.

A timing error occurs when the specification says one thing and the implementation does something else. Often times, this is caused by execution at a time that violates a specification.



Model-based design, simulation, and synthesis is being used more than before in lieu of hand writing code and testing it [4].

There is also a resurgence in Cyber Physical System design due to renewed interest in the area

If these trends continue, we will see:

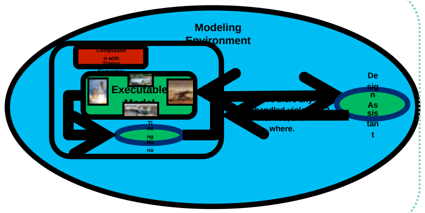
1. **More use of model-based design with timing specifications in the design of Cyber Physical Systems;**
2. **The desire to include error handling explicitly in a model instead of in an ad hoc manner.**

Objective

Add meaning to what is done in the event of a timing error. We achieve this by :

1) Extend concepts from real-time programming languages to model-based design.
 * Exception handling

2) Adding concepts to hierarchical state machines
 * Error Transitions



Timing Manager

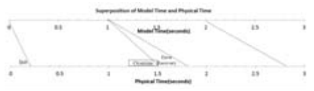
1. Notes desire to incorporate physical time into the model
2. Annotates actors with a execution time estimate parameter. Values are expected to be provided by the designer or by an external execution time analysis tool
3. Simulates execution time of actors as a probabilistic variant of annotated execution time estimate parameter value
4. If a timing error is detected the timing manager passes the error up the model hierarchy. If the specification is included in a modal model the timing manager enables the first applicable error transition. If not it moves further up the hierarchy and attempts to enable an error transition
5. If there is not error transition to catch the error in the hierarchy an exception the simulation of the model is stopped and the user is informed of the unhandled error

Discussion and Future Work

Error Handling in Model-Based design for Real-Time Systems is still a work in progress. Preliminary results indicate that the current mechanism is able to detect and appropriately transition after simulating a timing overrun.

Features Added to Ptolemy II

1. Error Transition for timing errors into modal models[3].
2. Timing manager to introduce a secondary notion of time and handle errors hierarchically.
3. Design Assistant to aid the user
4. Preliminary code generation support for the timing manager.



Future Work

2) incorporating representative probabilistic distributions into the timing
3) expanding the preliminary work in C and Java code generation
Allowing the user to specify a recovery transition
Adding in other types of timing error transitions

Code generation support for all features of the timing manager.

Timing manager extensions to provide suggestions for the scheduling strategy used with the MOC.

References

[1] ARIANE 5: Flight 501 Failure. http://sspg1.bnsc.rl.ac.uk/SEG/Ariane%205%20Explosion%20Report.htm
[2] Mars Spirit Wiki. Mars Spirit Software Problem. http://c2.com/cgi/wiki?MarsSpiritSoftwareProblem.
[3]C. J. Murray. Automakers opting for model-based design. Design News: http://www.designnews.com/article/511392-Automakers_Opting_for_Model_Based_Design.php, November 5 2010.
[4] Edward A. Lee, Stavros Tripakis. "Modal Models in Ptolemy". Proceedings of 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT 2010), 1-11, October, 2010.

Acknowledgements

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS:PRET) and #0720841 (CSR-CPS)), the U. S. Army Research Office (ARO#W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the State of California Micro Program, and the following companies: Agilent, Bosch, HSBC, Lockheed-Martin, National Instruments, and Toyota.

This work was also supported by the Jenkins Pre-doctoral Fellowship program as well as a Jenkins Pre-doctoral Fellowship Program Mini Grant