Towards Decentralized Management of Graceful Degradation in Distributed Embedded Systems

Osamah A. Rawashdeh
Electrical and Computer Engineering Department
Oakland University, Rochester, Michigan, USA
rawashd2@oakland.edu

Abstract

Graceful degradation entails a proportional loss of functionality or the reduction in the quality of services a system provides in response to faults. Compared to traditional techniques, graceful degradation is a promising approach to achieving fault tolerance at reduced cost. Current research using this approach in distributed embedded systems assumes a central management unit responsible for tracking resources and for system reconfiguration. Such units are single points of failure and maybe become costly when made fault tolerant by traditional techniques. This papers overviews current work on developing a framework that enables the specification and implementation of real-time distributed embedded systems that feature decentralized management.

1. Introduction

Fault tolerance is typically achieved through redundancy in hardware and software to enable fault detection and recovery. Explicit spatial redundancy for a non-trivial embedded system can however be complex and costly in terms of size, weight, price, and power consumption. These costs can be prohibitory in mass consumer products such as automobiles.

A promising new concept to achieving flexible fault tolerance at lower cost is to design systems that gracefully degrade in response to faults. A change in operating mode in response to faults that causes the loss of non-critical functionality or a reduction in the quality of the services a system provides is formally known as *graceful degradation* [1,2]. The goal of this work is to provide a framework for modeling and implementing adaptive real-time distributed embedded systems capable of graceful degradation.

This work has three principal goals. The first goal is to design systems without a central management unit, which is typically responsible for tracking resources and for system reconfiguration. Such central units are single points of failure and are costly when made fault tolerant by traditional techniques. The second goal is to provide a modeling tool that allows the analysis of all possible operating modes at design time. Such capability is essential in safety-critical applications, where all operating modes have to be explicitly verified before system deployment. Finally, specifying standard network interfaces, fault checking, and reporting standards for modular system components rather than providing resource hungry middle layer software.

2. Application Specification

An application is comprised of a set of *components* that are interconnected and cooperate over a network. A component is a combination of hardware and software that produce and/or consume data and has a standard network interface. There are three kinds of components: input, output, and processing components. Input components periodically *produce* estimates of physical quantities in form of data packets onto the system network. Output components, on the other hand, are *consumers* of data and drive system actuators (i.e., system services). Finally, processing components perform system-internal data processing and hence consume as well as produce data.

The data flow between components is captured graphically in a tree form similar to success trees [3]. The graphing technique allows a system architect the specification of different degrees of dependency on input data. For example, a component may have a set of required inputs in addition to a set of optional inputs. Data is processed as it become available on the system network. The quality of the data received and the availability of optional data affect the quality of the

data produced. The result will be a tree structure for each system output. The graph nodes are annotated with attributes, which include data sizes, criticality, and deadlines. These graphs also serve as a modeling tool for verification purposes.

3. Communication

A reliable TDMA broadcast network is assumed for transport of uniquely identified data packets between producer and consumer components. In a periodic system, the data size and their broadcast frequency as well as the total bandwidth available is known at design time allowing for proper bandwidth allocation. Worst case network latency, on the other hand, can be accounted for in the worst case execution time of components. A controller area network (CAN) will be used in first implementations of the proposed system.

4. Fault Tolerance

The framework is to support three types of fault tolerance techniques: masking fault tolerance, graceful degradation, and a fail-safe behavior. Masking fault tolerance can be achieved in two ways. It can be achieved by designing the system to include redundant producers of the same data. Secondly, a secondary standby source can be available in the system that detects the crash failure of the primary source in which case it starts providing the needed data for the consumer in a transparent fashion. Graceful degradation can be implemented by having secondary standby sources of data be of a lower quality than the primary producers. Also, non-critical output components can be implemented to recognize the absence of required input data and respond by shedding affected system functionality. Finally, a fault affecting a critical system output can be handled by failing safely. This can be achieved by having a special system wide error message be broadcasted on the network resulting in all system components switching to a predefined fail-safe mode.

5. Related Work

Graceful degradation in embedded system is an active research area. The Chameleon [4], Ardea [5], and RoSES [6] projects are relevant approaches that however include central management units. Jini [7] and CORBA [8], on the other hand, are two frameworks for designing truly distributed service-based systems but are designed primarily for large information systems and therefore are too resource demanding for the embedded systems targeted in this paper. The recently released CORBA/e [9] (e for embedded systems) is aimed at embedded and real-time applications but yet requires at least 32-bit processors. Finally, other research efforts focus on the modeling and verification of adaptive systems and do not consider runtime management matters in depth.

6. Conclusion

This paper briefly overviewed a modular service-based approach to designing distributed embedded systems capable of masking as well as graceful degradation in response to faults. The main goal of this work is to develop a set of design methodologies that result in a system that does not require centralized management and allow the testing of all possible configurations to make it suitable for safety critical applications.

7. References

[1] Herlihy, M.P., “Specifying Graceful Degradation,” IEEE Transactions on Parallel and Distributed Systems, vol. 2(1), pp. 93-104, January 1991.

[2] Koopman, P., “Elements of the self-healing system problem space,” Workshop on Architecting Dependable Systems (WADS03), May 2003.

[3] Modarres, M., “Functional Modeling of Complex Systems Using a GTST-MPLD Framework,” Proceedings of the International Workshop on Functional Modeling of Complex Technical Systems, May 1993.

[4] Kalbarczyk, Z.T., Iyer, R.K., Bagchi, S., and Whisnant, K., “Chameleon: A Software Infrastructure for Adaptive Fault Tolerance,” IEEE Transactions on Parallel and Distributed Systems, vol 10(6), pp. 560-579, June 1999.

[5] Rawashdeh, O., and Lumpp, J., “Run-Time Behavior of Ardea: A Dynamically Reconfiguring Distributed Embedded Control Architecture,” IEEE Aerospace Conference, IEEEAC Paper# 1516, March 2006.

[6] Shelton, C., Koopman, P. and Nace, W., "A framework for scalable analysis and design of system-wide graceful degradation in distributed embedded systems," WORDS 2003, January 2003.

[7] Jim Waldo, “The Jini Architecture for Network-Centric Computing,” Com. of the ACM, vol. 42(7), pp. 76-82, 1999.

[8] Markus Aleksy, Axel Korthaus, and Martin Schader, *Implementing Distributed Systems with Java and CORBA*, Springer, Heidelberg, 2005.

[9] Object Management Group, Inc., “CORBA/e Resource Page,” April 2008, http://www.omg.org/corba-e.