Distributed Real-Time Computing

Prof. John A. Stankovic
Department of Computer Science
University of Massachusetts
Amherst, Mass. 01003

Abstract
This invited position paper outlines the key areas of research in distributed real-time systems that are being investigated within the Spring Project at the University of Massachusetts. This includes reflective, multiprocessor operating systems, dynamic guarantees, adaptive flow control filters, adaptive fault tolerance, architecture support, and active real-time databases.

1 Introduction
Constructing distributed real-time systems is perhaps the most difficult problem for real-time systems designers. Such systems are normally involved with controlling large, complex, non-deterministic environments, yet predictability, and timing and fault tolerance guarantees are required. Today, many distributed real-time systems are highly static. These systems can be quite effective, but they tend to be inflexible and have difficulty if operating in large, complex, non-deterministic environments. Other distributed real-time systems are built in a seat of the pants manner. Such systems do not meet the predictability and guarantee requirements. What is required is a combination of engineering and scientific results to design, build, and analyze distributed real-time systems of the future. I believe that such results are a long way off. In the Spring Project at the University of Massachusetts techniques are being developed to move the state of the art towards predictability and analyzable guarantees for distributed real-time systems. Each of the following sections briefly identifies some of this work.

2 Reflective Operating Systems
In simplified terms, reflection can be defined as the computational process of reasoning about and acting upon the system itself. Reflection, as a concept, is not new and almost all systems have some reflective information in them. However, identifying reflection as a key architectural principle and exploiting it is very new, and this is what is done in the Spring kernel [5]. To best support the flexibility required for distributed real-time systems over the long lifetimes of systems, we believe that it is absolutely essential to have general architectural structures capable of holding current state and semantics of the application, as well as, the system primitives for changing that information, and general (changeable) on-line methods for using that information. In other words, such reflective information can be monitored, used in decision making during system operation, and easily changed thereby supporting many flexibility features. Much research is still required to identify what meta-level information is important, how to represent it, and to determine the performance implications including performance of the meta-level. In the Spring multiprocessing kernel we have made such decisions and are experimenting with them in a flexible manufacturing domain.

3 Dynamic Guarantee Algorithms
We have developed several versions of guarantee algorithms [3] which provide different levels of guaranteed service for hard real-time systems in the LAN environment. We have implemented these algorithms both in software [5] and in VLSI [2] (a co-processor). The paradigm presented by the Spring scheduling algorithms and kernel include:
- specification of quality of service,
- algorithmic and kernel support to actually achieve this guaranteed service at runtime,
- reservations of sets of resources in an integrated fashion,
- end-to-end scheduling, and
- atomic guarantees for sets of tasks.

4 Adaptive Flow Control Filters
Using adaptive flow control filters that we are developing [6], the overall view of a distributed real-time system consists of many sensors whose values are input to a set of adjustable filters. These inputs may occur in an unpredictable manner and even arrive at rates or total amounts beyond original specifications. The filters then create controlled inputs to the rest of the system in a manner so as to avoid catastrophic failure and so that they are integrated with timing constraints in the rest of the system in a predictable manner.

5 Adaptive Fault Tolerance
One key aspect of distributed real-time systems lies at the intersection of adaptive fault tolerance and real-time computing. Our goal is to combine adaptive management of redundancy [1] and adaptive real-time scheduling to allow better exploitation of resources in the absence of failures and overloads, and better
adaptive degradation in their presence. We are investigating research questions in the (1) specification, (2) analysis and (3) runtime implementation support for adaptive fault tolerance under real-time constraints. We recognize that trying to cover all three areas is demanding, but it is critical to address the issues in an integrated fashion. For example, if we permit too much generality in the specification it may be impossible to analyze or be too costly to implement.

We have developed a notation for specifying adaptive fault tolerance with timing constraints called FERT [1]. FERT allows a designer to treat each FERT object as a fault tolerant entity with protection boundaries (fault containment units), initially without worrying about timing and redundancy issues. The designer specifies a set of application modules as part of a single entity. The application modules represent user code for redundant operations, voters, or more general adjudication code. Each of these application modules would be written in some real-time language with associated timing requirements. In addition to application modules, the FERT designer also supplies one or more adaptive control policies using a set of primitives. These primitives explicitly interact with scheduling and analysis algorithms. Timing constraints, task value, and other task characteristics can be input to the FERTs and used by the adaptive control strategies.

6 Architecture Support

We are investigating a number of hardware-software based ideas for moving distributed real-time systems towards predictability, i.e., to provide support for predictable systems even though the environment and various components may be unpredictable. These ideas are: scheduling co-processors used for dynamic planning (Section 3), adjustable flow control filters (Section 4), reflective memories, and ATM switches.

Reflective memories are commercial products used in many tightly coupled distributed real-time systems to avoid the unpredictable aspects of most standard IPC mechanisms. However, published descriptions of how they are used are limited to having nodes interrupt each other and sharing input and output data and control. We are studying their use in supporting distributed shared memory for loosely coupled distributed real-time systems. We have already implemented distributed shared memory in the Spring kernel using these reflective memories. We are now experimenting with this system to investigate issues such as:

- how much more efficient and predictable is distributed shared memory than typical IPC primitives,
- where are the bottlenecks in the implementation of distributed shared memory on reflective memories and should a different hardware-software solution be proposed,
- how should distributed synchronization be supported,
- how does distributed shared memory contribute to solutions for dynamic real-time systems, and
- how would use of reflective memories compare with the use of ATM switches for predictable distributed communication.

7 Active Real-Time Databases

In large, distributed real-time systems, data spans the gamut from simple control variables and sensor values to large and complicated relations typically found in database systems. This data often has a temporal validity interval, inconsistencies in the data may trigger subsequent actions, and transactions which use the data may have deadlines. To deal with this complicated situation in a unified manner we are investigating the use of an active, temporal, real-time database system. We have implemented RADAex, a simulator for this type of database system and are experimenting with database and OS protocols and algorithms needed to support real-time databases [4].

References


