A Resource Management Model for Dynamic, Scalable, Dependable, Real-Time Systems

Binoy Ravindran, Lonnie R. Welch, Carl Bruggeman, Behrooz A. Shirazi, and Charles Cavanaugh*

The University of Texas at Arlington

Abstract. Dynamic real-time systems function in unpredictable environments and have requirements that span many domains such as time, survivability, and scalability. The system requirements are typically determined as a function of the environment, further exacerbating the unpredictability of the problem. Existing solutions, for the most part, have focussed on problems for which the attributes are static, and there exists a rich set of solutions for such problems. Our problem domain has attributes that are inherently dynamic rather than static, requiring a new approach.

We present a model for resource management for dynamic real-time systems. The model includes a language for specifying both static and dynamic attributes. Dynamic attributes include those that are measured as well as requirements that are evaluated on-line. In addition, we describe a resource management architecture for such systems.

1 Introduction

Computer-based control systems are used to control physical environments with the aim of maintaining the environment in a certain desired state. The system accomplishes this task by continuously assessing the environment, performing corrective actions when necessary (causing changes in the state of the environment), and providing continuous guidance for the result of those actions until the actions have accomplished their goals. Such systems are said to be *real-time* when the system must react to changes in the environment in a timely manner in order to ensure correct behavior of the system. Furthermore, for systems that function in critical environments, continuous *availability*, and the ability to *scale* to satisfy the changing processing demands (in a timely manner) are also major concerns.

Real-time control systems have several important characteristics that profoundly influence resource management design and implementation. They include the behavior of events that prompt an action from the control system,

* Department of Computer Science and Engineering, The University of Texas at Arlington, Box 19015, Arlington, Texas 76019-0015, E-mail: binoy@cse.uta.edu.
Sponsored in part by DARPA/NCCOSC contract N66001-97-C-8250, and by the NSWC/NCEE contract NCEE/A303/41E-96.

the size of the data stream that the system has to process for decision making, and types of timeliness requirements. Events from the environment have a temporal behavior in the sense that they may occur in a *periodic*, *aperiodic*, or *transient-periodic* manner. Periodic events are those that occur repetitively with a period that may remain constant or may change with time. Aperiodic events on the other hand, occur at arbitrary points in time in a non-repetitive manner. A characterization of the inter-arrival times of the aperiodic events may or may not be possible. A lower bound on the inter-arrival times between the (successive) occurrences of the aperiodic event may be known or may change with time. Transient-periodic events are events that occur at arbitrary points in time. However, once the event occurs, it repeats with a period for a certain duration of time. This period may remain constant or may change during each occurrence of the event. Also, a lower bound on the inter-arrival times between the (successive) occurrences of the transient-periodic event may be known or may change with time.

Another factor that strongly impacts the design of resource management algorithms is the size of the data stream which the systems have to process for decision making and reaction. Reaction of the system to events from the environment is based on the processing and evaluation of data collected from the environment. The size of this data stream may remain a constant or may vary by orders of magnitude with each collection.

Timeliness requirements are another important type of user requirement. Typical timing requirements include *deadlines*, *throughputs*, and *datum inter-processing times*. Deadlines are maximum time deadline requirements on the response times of control system functions. The throughput requirement, requires the control system to process data items collected from the environment within a certain amount of time. Inter-processing time is a maximum time deadline requirement on time delays *between* datum processings' of the control system [2]. The timeliness requirements of the system may be determined off-line or may be evaluated on-line.

The literature on real-time systems provides models and algorithms for real-time problems that have mostly static characteristics. Examples include [2], [3], [1], [9], [4], [8]. For a discussion on this issue, please see [6]. For the dynamic class of real-time systems, properties of control system functions, such as periods (of both periodic and transient-periodic events), bounds on inter-arrival times, and data stream sizes, change with time. Furthermore, most system timeliness requirements also change dynamically.

Some of the efforts toward addressing the class of real-time problems that function in dynamic environments include [5], [7], and [10]. [7] present models and algorithms for dynamic problems that is based on static system profiles and dynamic measurements. A resource management model for systems with multiple applications, each of which have several Quality-of-Service (QoS) requirements is described in [5].

[2] For a detailed discussion on the different types of timeliness requirements, please see [6].

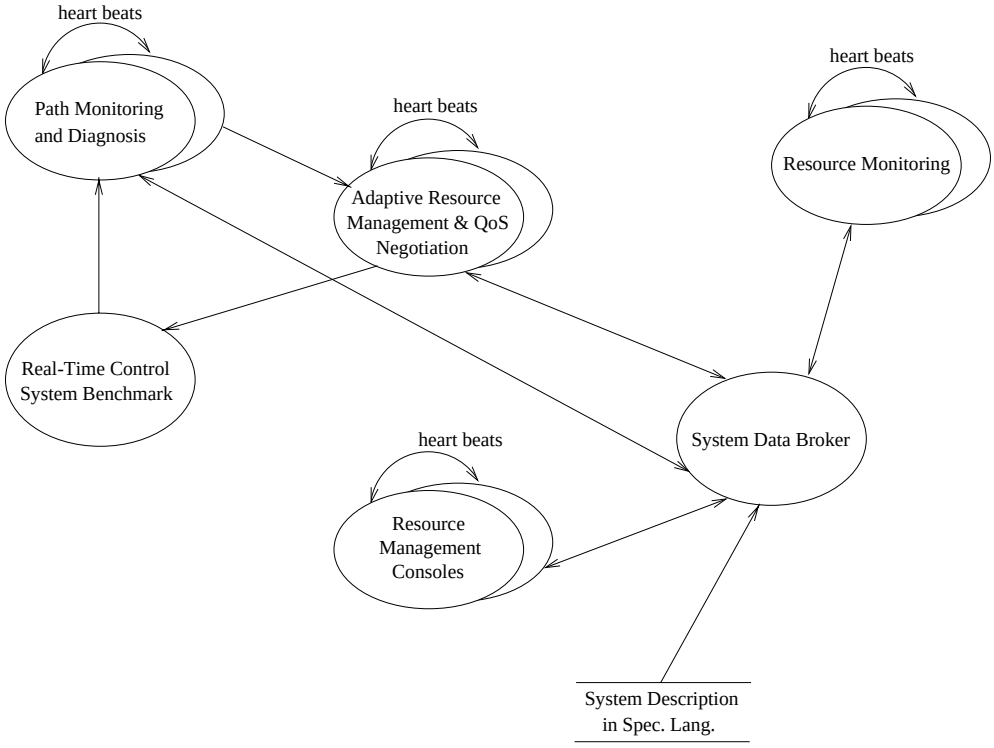

Fig. 1. The Resource Management Architecture

Our resource management model is part of the DeSiDeRaTa project at the University of Texas at Arlington. This project involves building middleware services for the next generation of ship-board air defense systems being developed by the U.S. Navy.

The rest of the paper is organized as follows. The resource management architecture is presented in Section 2. The resource management model is described in Section 3. The specification language is discussed in Section 4.

2 The Resource Management Architecture

The resource management architecture (Figure 1) consists of components for adaptive resource management and QoS negotiation, data broker, path monitoring and diagnosis, resource monitoring, and resource management consoles.

The adaptive resource management and QoS negotiation component is responsible for making resource management decisions. This component computes the allocation decision by interacting with the data broker and obtaining software and hardware system profiles. The allocation decision may involve migrating programs to different hosts nodes, starting additional copies of programs (for

scalability), or restarting failed programs (for survivability). The resource management component carries out its decisions by communicating with a daemon program (on each host) to start up and control programs on each host.

The data broker component is responsible for collecting and maintaining all system information . The data broker reads the system description and requirements expressed using the specification language, and builds the data structures that model the system. Dynamically measured software performance metrics, such as path latency and throughput, and resource usage characteristics, such as program page faults and resident size, are collected and maintained by the path monitoring and diagnosis component. The data broker obtains measurements of the dynamic attributes of the software from the monitoring component. Hardware resource profiles are collected and maintained by the resource monitoring component, and fed to the data broker on demand as well as periodically. The data broker thus provides a single interface for all system data.

The path monitoring and diagnosis component monitors the performance of software systems at the path-level[3]. This component determines the changing requirements of the software by interacting with the data broker. When a path fails to meet the requirements, this component performs diagnosis of the path, and determines the "bottleneck" node of the path.

Resource management consoles display system and path status, and allow dynamic attributes, such as deadlines, to be modified. All communication for such consoles is through the data broker.

Survivability in the resource management architecture is achieved by replicating most system components. Replicas periodically sends "heart-beat" messages among themselves to check on liveliness.

3 The Resource Management Model

The resource management model explicitly divides system attributes into ones that are static and dynamic. Static attributes are attributes whose values are known off-line. This includes both user specified requirements and properties of system architecture on which the system is running. Examples of statically known system element properties include, types of nodes at different levels such as path types (software level), LAN types (hardware level), and host types (hardware level). Dynamic attributes are either measured attributes that change during execution or attributes, such as requirements, whose value cannot be determined off-line. Examples include: path latencies (software level), program execution times (software level), LAN throughputs (hardware level), host utilizations (hardware level), subsystem/path priorities, simple-deadlines, and the number of additional activations (of paths). A partial set of model attributes is shown in table 1.

[3] In our air defense system example, nearly all requirements are expressed at the path level.

Table 1. A partial set of model attributes

System element (node)	Software Attribute		Hardware Attribute		Mapping Attribute	
	Static	Dynamic	Static	Dynamic	Static	Dynamic
subsystem-priority		√				
path-type	√					
path-priority		√				
path-simple-deadline		√				
path-survivability	√					
path-super-period-deadline		√				
path-latency		√				
program-startup-constraint	√					
program-user-time		√				
program-page-faults		√				
program-resident-size		√				
task-priority		√				
LAN-type			√			
LAN-zone			√			
LAN-obsvd-throughput				√		
host-type			√			
host-speed			√			
host-zone			√			
host-free-memory				√		
host-idle-percentage				√		
program-path					√	
program-arguments					√	

4 The Specification Language

One system component required by the data broker is a description of the system software and hardware components. In our system, the characteristics of a real-time control system are specified using a declarative language. The language consists of three parts, a description of the software components, a description of the hardware components, and a description of all of the allowable mappings for each of the software components to the hardware components. The language provides a number of built in data types that are used to specify the characteristics of the system components. These characteristics include software requirements such as deadlines, throughput, and interprocessing time; hardware related data types such as number of processors, processor speed, and network bandwidth; and mapping related data types, such as the location of the executable image on a particular host.

Software systems are viewed as a directed graph of software nodes where an edge exists between two software nodes if there is a flow of data from one software node to the other. This graph of software components is described in a hierarchical manner. A node at a particular level is described in terms of a

graph of nodes at the next level. Software characteristics can be attached to nodes at any level. Nodes at the lowest level (terminals) are distinguished by having only characteristics. The number of levels and the names of each level must be specified by the user. In our air-defense system example there are four software levels: *subsystem*, *path*, *program*, and *task*.

Hardware systems are viewed in a similar fashion as a graph of hardware nodes where an edge exists between two hardware nodes if data can flow from one hardware node to the other. The hardware graph is described in a hierarchical manner as well, in which a node at a particular level is described in terms of a graph of node at the next level. In our air-defense system example there are two hardware levels: *LANs* and *hosts*. Interconnecting devices, such as bridges and routers are considered hosts because many hosts have capabilities for routing that the resource management algorithm can utilize.

The language also provides a mechanism for specifying the mapping constraints for the software component onto the hardware components. The mapping allows the executable program name and command line arguments to be specified for every host and LAN pair that a software component has been configured to use. These mappings represent additional constraints the resource management algorithm must consider for resource reallocation decisions.

References

1. E. D. Jensen and J. D. Northcutt. Alpha: A non-proprietary operating system for large, complex, distributed real-time systems. In *Proceedings of The IEEE Workshop on Experimental Distributed Systems*, pages 35–41, Huntsville, AL, 1990.
2. C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
3. C. D. Locke. Software architecture for hard real-time applications: Cyclic executives vs. fixed priority executives. *Real-Time Systems*, 4(1):37–53, March 1992.
4. A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment.* PhD thesis, M.I.T, Cambridge, MA 02139, 1983.
5. R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for qos management. In *Proceedings of The 18th IEEE Real-Time Systems Symposium*, pages 298–307, 1997.
6. B. Ravindran and L. R. Welch. A taxonomy of real-time systems. Technical Report TR-CSE-97-002, The University of Texas at Arlington, April 1997.
7. D. Rosu, K. Schwan, S. Yalamanchili, and R. Jha. On adaptive resource allocation for complex real-time applications. In *Proceedings of The 18th IEEE Real-Time Systems Symposium*, pages 320–329, December 1997.
8. S. Sommer and J. Potter. Operating system extensions for dynamic real-time applications. In *Proceedings of The IEEE Real-Time Systems Symposium*, pages 45–50, December 1996.
9. J. P. C. Verhoosel. *Pre-Run-Time Scheduling of Distributed Real-Time Systems: Models and Algorithms.* PhD thesis, Eindhoven University of Technology, The Netherlands, January 1995.
10. L. R. Welch et al. Challenges in engineering distributed shipboard control systems. In *Proceedings of The Work-In-Progress Session*, December 1996. The 17th IEEE Real-Time Systems Symposium.