A Distributed Architecture for QoS Management of Dynamic, Scalable, Dependable, Real-Time Systems[1]

Lonnie R. Welch and Behrooz A.Shirazi
Computer Science and Engineering Dept.
The University of Texas at Arlington
Arlington, Texas 76019-0015

Abstract

Many distributed real-time systems of the emerging generation have rigorous Quality of Service (QoS) objectives. They must behave in a dependable manner, respond to threats in a timely fashion, and provide continuous availability within environments that are hostile and unpredictable. Furthermore, resources should be utilized in an efficient manner, and scalability must be provided to address the ever-increasing complexity of scenarios that confront such systems. This paper describes the DeSiDeRaTa project, which is addressing the needs of such systems. DeSiDeRaTa research efforts are focusing on: i) QoS specification, ii) QoS metrics, iii) distributed middleware for dynamic QoS management, and iv) benchmarking. This paper provides an overview of the DeSiDeRaTa middleware architecture.

Key words and phrases: dynamic real-time systems, QoS specification, QoS metrics, dynamic QoS management, survivability, scalability.

[1] Sponsored in part by DARPA/NCCOSC contract N66001-97-C-8250, and by the NSWC/NCEE contract NCEE/A303/41E-96.

1. Introduction

The DeSiDeRaTa[2] project is providing innovative QoS management technology which includes a specification language and dynamic QoS management software that support *dynamic, path-based systems* [5,6,7]. "Dynamic paths" have proven useful for manual QoS assessment and resource allocation during the engineering of the Navy's distributed AAW prototype within the HiPer-D testbed. A dynamic path may consist of sensors, actuators, and control software for filtering, evaluating, and acting. The paths may have timing constraints, may have widely varying (an even unpredictable) dynamic behavior, may be scalable and may be fault tolerant.

Most previous work in distributed real-time systems has focused on a lower level of abstraction than the path and has assumed that all system behavior follows a statically known pattern [3,4]. Scalability, efficient use of computing resources, and survivability can become problems when applying the previous work to some large applications [1]; furthermore, it is sometimes impossible to obtain some of the parameters required by the models! In contrast, the large granularity of the dynamic path accommodates such large systems; the dynamic properties of a path allow the modeling of systems that work in environments that have unknown scenarios; and the dynamic path paradigm is based on obtainable parameters, since it evolved from the study of existing computer systems.

This paper describes the technical approach of the DeSiDeRaTa project. Section 2 provides an overview of a dynamic QoS management architecture which supports dynamic path-based systems. Section 3 describes the DeSiDeRaTa approach to adaptive resource allocation. An architectural view of our approach to monitoring and instrumentation is described in Section 4.

2. Dynamic QoS Management Architecture

Fig. 1 depicts the DeSiDeRaTa architecture for dynamic QoS management. Primary components of the architecture are *the path-based real-time control system* (shown in the figure as RTCS/BM), *path monitoring and diagnosis, adaptive resource management and QoS negotiation, resource management console(s), resource monitoring, system data broker* and *system description.*

Path monitoring involves the collection of data regarding (1) the real-time performance of paths and subpaths (a subpath is a program or a communication between programs), and (2) the resource utilization of subpaths. Additionally, path monitoring determines when QoS of a path becomes (or is likely to become) undesirable. The subpath(s) that is causing poor path QoS is (are) identified by the *path diagnosis* component.

---

[2] The name DeSiDeRaTa was chosen to represent the class of **D**ynamic, **S**calable, **D**ependable, **R**eal-**T**ime computing systems.

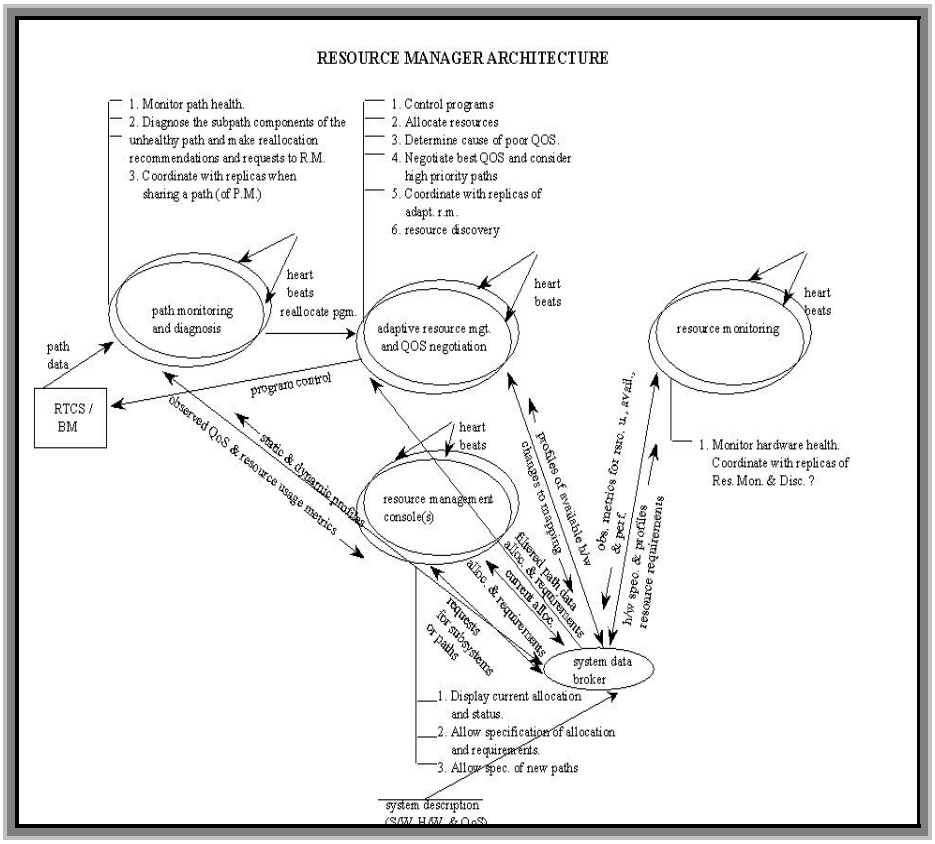

Fig. 1. DeSiDeRaTa architecture for dynamic QoS management.

Adaptive resource management automatically adjusts the allocation of resources to control *path-level QoS* (when poor QoS is detected). QoS negotiation performs tradeoff analysis among requirements to perform allocation selection under overloaded conditions wherein insufficient hardware resources are available to accommodate all system QoS requirements. *Resource management consoles* provide hierarchical views of the real-time control software, the distributed hardware, and the mapping (of software to hardware); these views depict structure as well as monitoring data. The consoles also permit manual and semi-automatic resource management.

Resource monitoring capabilities provide application profiles and instrumentation data. Application profiles include hardware resource requirements for particular QoS levels (e.g., CPU, memory and network resources required to provide the required timeliness QoS to a single activation of an *engage* path of an AAW system, or for the AAW *detect* path to process a particular number of tracks within the desired latency). Instrumentation data includes dynamically updated information regarding both host

and network resource usage levels. The system engineer can interact with the instrumentation capability, toggling specific instrumentation capabilities and controlling instrumentation parameters (e.g., frequency and logging).

The *system data broker* collects data from the path and resource monitoring components and allows clients to access the data.

The *system description* represents the software and hardware characteristics, the requirements of the software, and the constraints on mappings of software to hardware.

3. Adaptive Resource Allocation

Adaptive resource allocation is performed in response to events such as: path overload (violation of real-time QoS requirements of a path), path under-load (over-allocation of resources to one or more paths or subpaths), failure or overload of a host or network resource, and death of a program. These events are detected by path monitoring and resource monitoring components, which notify the adaptive resource manager. Upon notification, the resource manager performs the following steps to bring the system back into an acceptable state:

1. Perform system diagnosis to determine a set of possible actions to restore QoS to required levels. Possible actions include moving a program, replicating a program, killing a program, and re-routing a communication subpath. Rank the set of actions according to their likelihood of restoring the system to an acceptable state.
2. Select a subset of the actions identified in step 1. The set of actions should restore the system to an acceptable state, and they should satisfy several objectives (such as minimizing the number of program movements and maximizing the slack available to real-time applications).
3. Perform the set of actions selected in step 2.

4. Monitoring Architecture

Adaptive resource management relies on the ability to dynamically monitor performance of distributed real-time applications, to project performance in the future, to monitor loading levels of all compute and communication resources, and to predict future loading levels. This section presents the architecture of the monitoring infrastructure of the DeSiDeRaTa middleware. As shown in Fig. 2, monitoring is performed for hosts, networks, paths, programs and middleware programs. Information repositories for each of the classes of monitored entities are maintained, as well a system-level information repository.

Fig. 3 depicts the architecture of the host monitors and shows the kinds of information collected on each host. Also, the figure shows the host monitor console, through which the characteristic of host monitors can be changed and logging can be enabled and disabled. The statistics collected for the host data are depicted in the lower right of the figure.

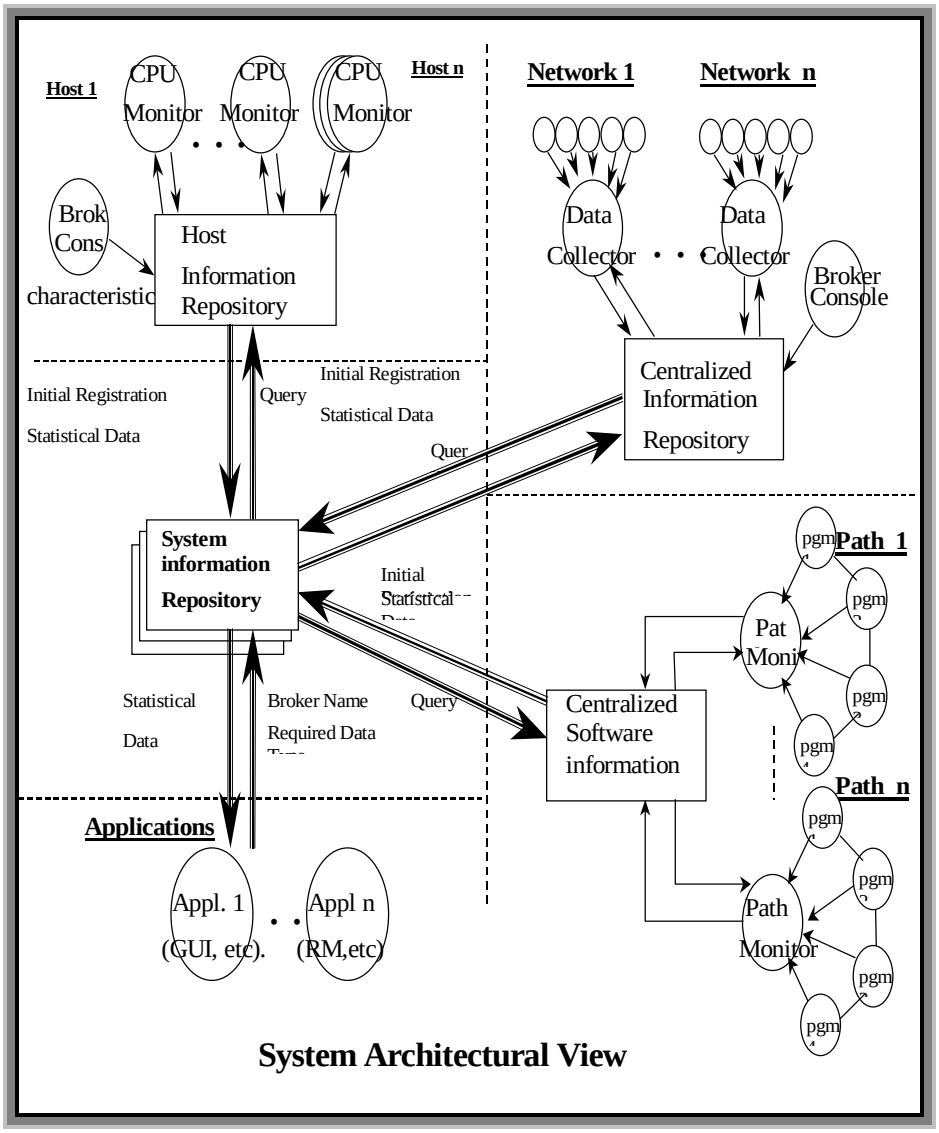

Fig. 2. System Information Repository Architecture

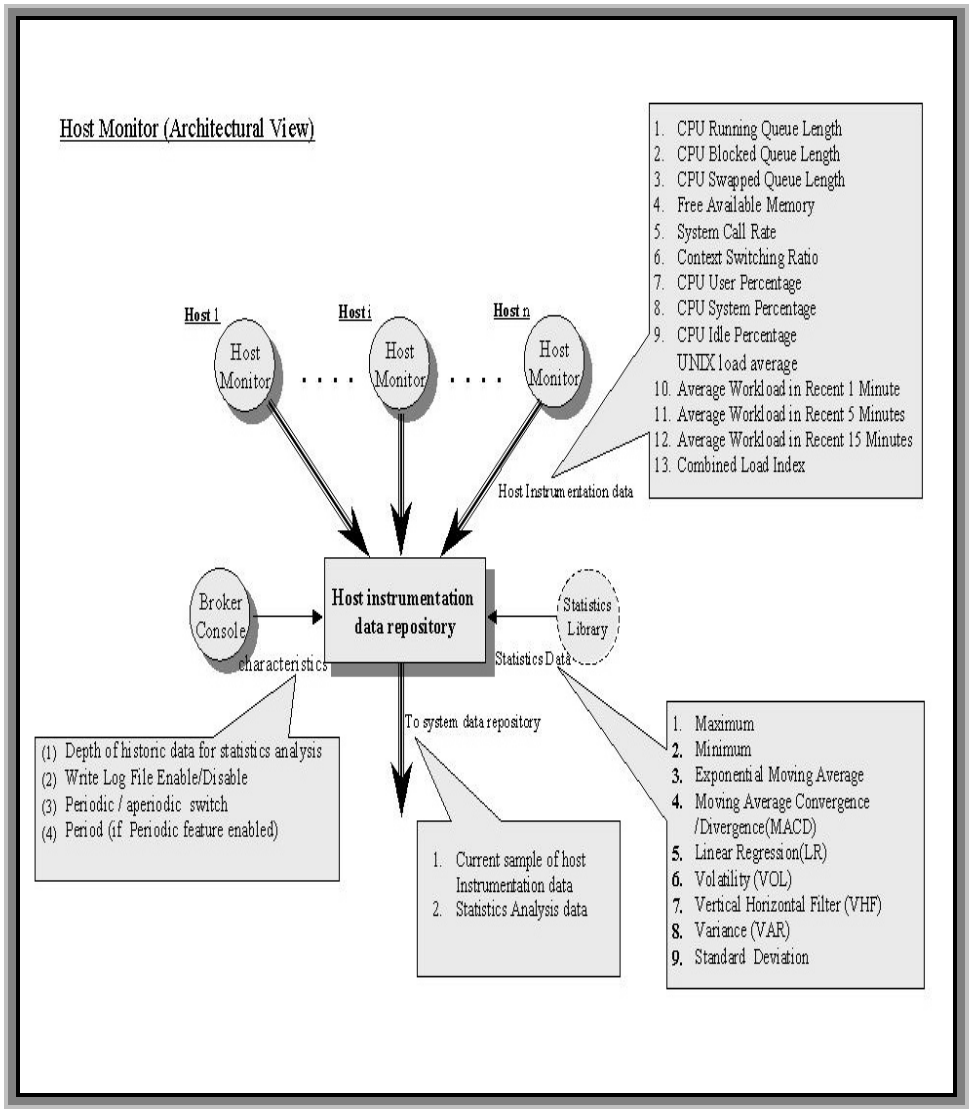

Fig. 3. Host Monitor Architecture

The network monitoring architecture is shown in Fig. 4. The approach taken for network monitoring is to sample the network loading between pairs of hosts and to use the samples to calculate expected latencies and throughputs between host pairs. This is achieved by a sender (client) and an echo server, as shown in the Fig. 4.

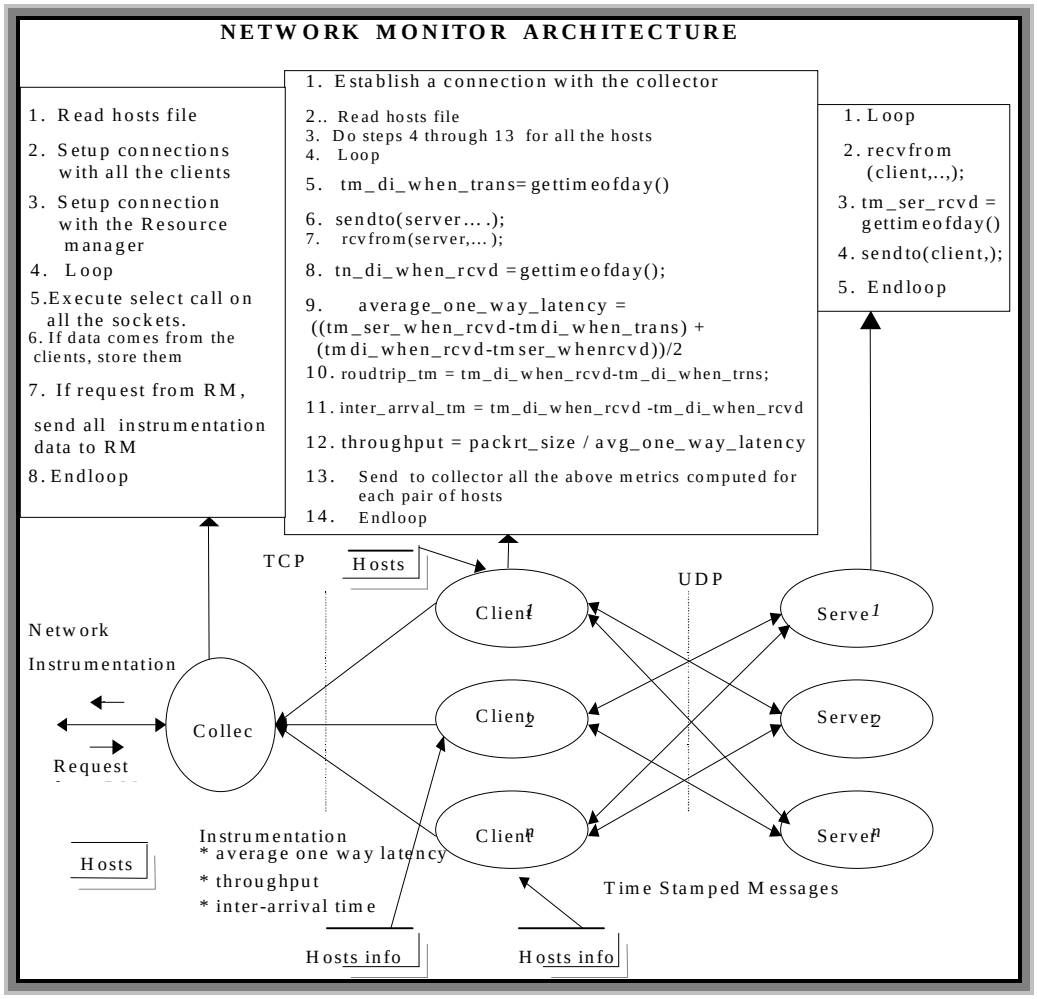

Fig. 4. Network Monitor Architecture.

Monitoring of programs is also performed, enabling the construction of application and path profiles. These profiles are used to build system models used in making resource allocation decisions. The actual data collected for each program is indicated in the lower right portion of the Fig.5.

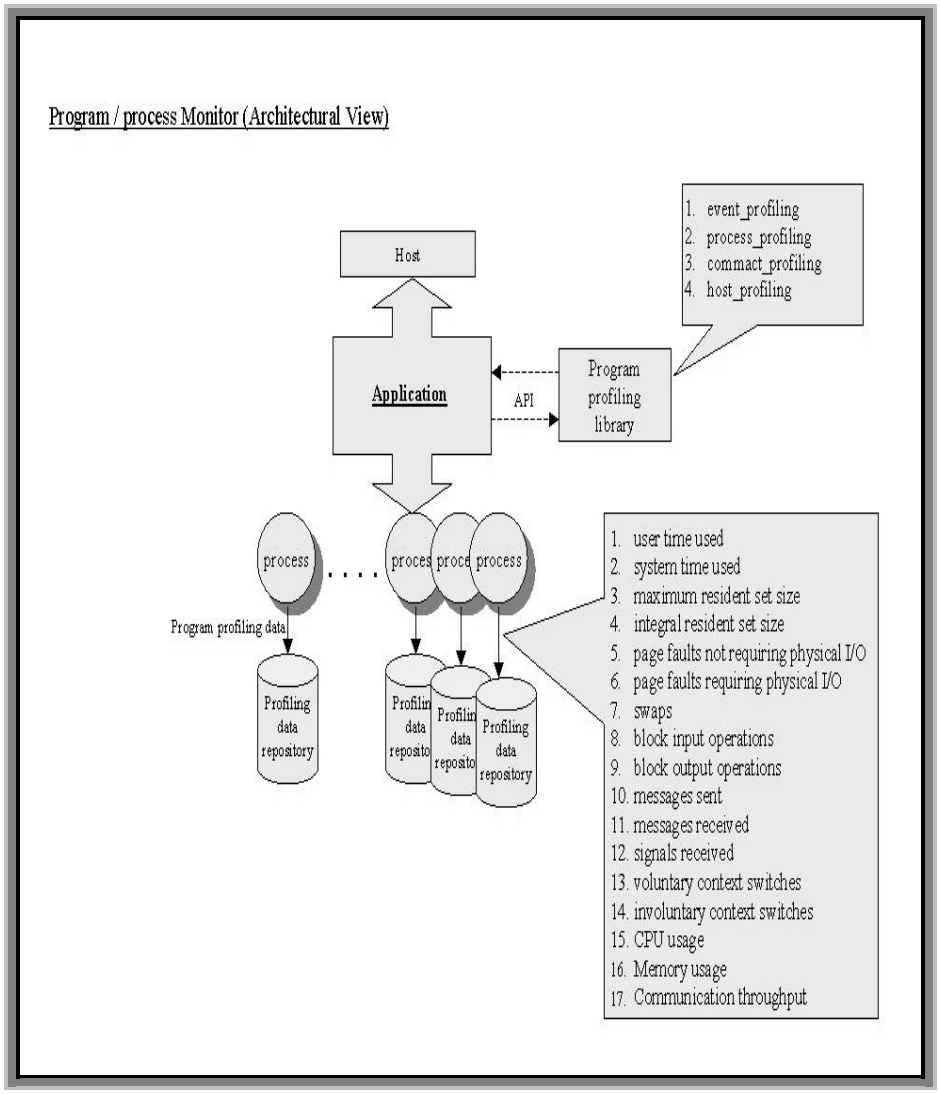

Fig. 5. Program Monitor Architecture.

5. Conclusions

DeSiDeRaTa provides INTEGRATED solutions for fault tolerant, distributed, dynamic real-time systems. Furthermore, the system model on which the systems engineering toolset is based differs significantly from that used in related work; it is at

a higher granularity, is based on instrumentation data and program structure, and it *combines* properties pertinent to dynamic, distributed, fault tolerant real-time systems.

6. References

[1] Gary Koob, “Quorum,” Proceedings of the DARPA ITO General PI Meeting, pages A-59 to A-87, October 1996.

[2] B. Shirazi, A.R. Hurson, and K. Kavi, “Scheduling and Load Balancing in Parallel and Distributed Systems,” IEEE Press, 1995.

[3] S. Son, "Advances in Real-Time Systems," Prentice Hall, 1995.

[4] J. Stankovic, and K. Ramamritham, "Advances in Real-Time Systems," IEEE Computer Society Press, April 1992.

[5] L. R. Welch, B. Ravindran, R. Harrison, L. Madden, M. Masters, and W. Mills, “Challenges in Engineering Distributed Shipboard Control Systems,” Work-in-Progress Session of The IEEE Real-Time Systems Symposium, December 1996.

[6] L. R. Welch and B. A. Shirazi, “DeSiDeRaTa: QoS Management Tools for Dynamic, Scalable, Dependable, Real-Time Systems,” Proceedings of the IEEE Workshop on Middleware for Distributed Real-time Systems and Services, 164-178, Dec. 1997.

[7] DeSiDeRaTa project website: http://zeus.uta.edu/~qos_prj