

Real-Time Adaptive Resource Management

Allalaghatta Pavan, Rakesh Jha, Lee Graba, Saul Cooper, Ionut Cardei, Mihaela Cardei, Vipin Gopal, Sanjay Parthasarathy, and Saad Bedros

Honeywell Laboratories

Distributed mission-critical environments employ a mixture of hard and soft real-time applications that usually expect a guaranteed range of quality of service (QoS). These applications have different levels of criticality and varied structures ranging from periodic independent tasks to distributed pipelines or event-driven modules.

The underlying distributed system must evolve and adapt to the high variability in resource demands that competing applications impose. The current industry trend is to use commercial off-the-shelf (COTS) hardware and software components to build distributed environments for mission-critical applications. Adding a middleware layer above the COTS components facilitates consistent management of system resources, decreases system complexity, and reduces development costs.

MIDDLEWARE COMPONENTS

The real-time adaptive resource management system consists of a middleware layer that provides integrated services for real-time mission-critical distributed applications. RTARM includes a number of features that facilitate distributed resource management:

- scalable end-to-end criticality-based QoS contract negotiation, which allows distributed applications to

share common resources while maximizing their use and execution quality;

- end-to-end QoS adaptation that dynamically adjusts resource use according to availability;



Adding a middleware layer above the COTS components facilitates system resources management, decreases complexity, and reduces development costs.

- integrated services for CPU, network, and other resources with end-to-end QoS guarantees;
- real-time application QoS monitoring for integrated services; and
- plug-and-play components for easy extensibility for new services.

As Figure 1 shows, RTARM's hierarchical architecture includes service managers—recursive structural entities that encapsulate a set of resources and their management mechanism. A higher-level service manager may receive a request for an integrated service that requires resources from lower-level service managers. Lower-level service managers directly control resources such as the CPU, network, or memory. The entire

hierarchy treats resources and negotiation requests uniformly. This hierarchy allows dynamic configuration because new service managers can join the system at any time.

The admission protocol builds a virtual spanning tree over the service manager hierarchy that remains valid throughout the application's lifetime. The service manager hierarchy forms a directed acyclic graph, with service managers as nodes and edges represented by the “uses-services-from” relation. The virtual spanning tree built over the service manager hierarchy allows the service managers to conduct QoS translation and reverse translation.

A hierarchical, recursive resource management architecture facilitates implementing QoS representations on top of basic services in complex distributed applications. A richer QoS representation simplifies application design and facilitates consistent resource management for incompatible applications. Regardless of the complexity of the appli-

cation architecture and the QoS semantics at the top of the service manager hierarchy, at the bottom of the hierarchy, everything translates to QoS requests for basic services. RTARM's hierarchical architecture scales well with large distributed environments. Service localization groups service managers into clusters to avoid communication bottlenecks. Sharing resources between lower-level and higher-level service managers adds redundancy, fault tolerance, and load balancing.

QoS TRANSLATION AND ADAPTATION

The ability of a mission-critical application to interact with its dynamic environment directly affects its performance. RTARM supports a multidimensional

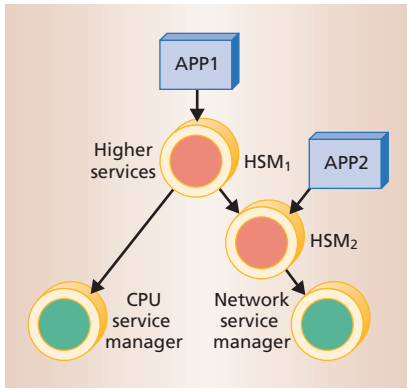


Figure 1. A simple RTARM hierarchy in which two applications request services from HSM1 and HSM2, the two higher-level service managers, and consume CPU and network resources.

QoS representation in which each dimension specifies an acceptable range $[Q_{min}, Q_{max}]$ for an application’s quality parameter. A set of range specifications, one per dimension, defines a QoS region. This QoS model facilitates resource negotiation and allows flexible resource management.

In RTARM’s recursive hierarchy, the QoS representation reflects the type of services a service manager provides. A higher-level service manager translates a QoS request for integrated services into the individual QoS requests that it and its lower-level service managers provide. When a service manager receives replies from its lower-level service managers, a QoS reverse translation process reassembles the returned request into its own QoS representation.

RTARM recognizes three situations in which an application QoS can change after admission:

- QoS contraction of lower-criticality applications when a new application arrives;
- QoS expansion when applications depart and release resources; and
- feedback adaptation, which varies the application’s current operational point within its contracted QoS region.

While the first two situations imply contract changes and involve other applica-

tions, feedback adaptation does not change the contract. Feedback adaptation relies on monitoring delivered QoS and uses the difference between delivered and desired QoS to adapt application behavior.

RESOURCE ALLOCATION

To achieve admission control and adaptation, RTARM applies a transaction-based two-phase commit protocol recursively at each service manager. The first phase addresses resource reservation. This phase executes a service availability test originating from the initiator service manager that received the admission request. The process moves down the spanning tree that resulted from the QoS translation process. The available reserved QoS undergoes reverse translation as it propagates back up to the initiator service manager from the lowest service manager layer.

In the second phase, the initiator service manager assesses the success status in the reservation phase and commits or aborts the transaction, which in turn commits or cancels service reservations along the spanning tree. If the system cannot allocate enough resources, a service manager adapts lower criticality applications to their minimum contracted QoS and uses the released resources for the new application.

Later, when resources become available, the service manager expands the QoS for the more critical applications. Sometimes, to admit a new, more critical application, the service manager squeezes the QoS of only a part of an existing distributed application. Changes in the high-level QoS may require collateral adaptation of other components that do not directly impact admission of the new application. For example, for a multimedia stream that has frame rate as a QoS parameter, if the service manager adapts one processing stage to the minimum rate, then all other stages will run at the same low rate.

ARCHITECTURE AND IMPLEMENTATION

RTARM provides a uniform internal architecture and common interfaces for all service managers. RTARM’s com-

mon object-oriented execution framework allows users to dynamically load system manager components from shared libraries during runtime configuration. A configuration manager passes information extracted from a configuration file to the system manager components during their initialization. A single executable program contains the empty system manager framework and configuration manager. The configuration manager loads specialized components from shared libraries to start the system managers.

We use this technique to apply specific Windows NT DLL components to initialize the CPU, network, and higher-level service managers. The flexibility of this plug-and-play feature permits implementation of a new service manager by just replacing a set of components for a particular interface, without rewriting the entire program.

APPLICATION SCENARIO

We tested and evaluated RTARM’s resource allocation and adaptation mechanisms on an automatic target recognition application. This mission-critical distributed, pipelined application schematically processes video images that a forward-looking infrared camera captures and then displays the recognized targets.

ATR processes the video images in seven stages. In stage 0, a sensor generates frames and then passes them through a series of filters and processing elements up through stage 6, which displays the identified targets. The frames are 8-bit gray-scale images with up to 512×512 pixels, and they contain from three to 50 targets. Some stages in the ATR application experience a highly variable workload, proportional to the number of targets.

RTARM features such as QoS expansion and contraction and feedback adaptation prevent queue accumulations, which have a negative effect on the end-to-end frame latency. For example, RTARM can tightly bound the ATR application’s end-to-end latency within a threshold of 10.8 seconds. Without RTARM’s hierarchical feedback adaptation mechanism, this latency continues to

grow unbounded. Using ATR, engaging weapons requires identifying targets within a prescribed deadline in typical suppression of enemy air defense scenarios.

We have successfully integrated a real-time communication toolkit as a network service manager in the RTARM hierarchy, and future plans include porting RTARM to a real-time Corba implementation to optimize its performance.

RTARM's feedback adaptation mechanism employs a simple control technique that causes wide oscillations at the operational point in the application's contracted QoS region. We are currently investigating classic control theory and application techniques such as proportional-integral-derivative control and multimodel predictive control to provide more predictable and stable feedback adaptation. *

Rakesh Jha is the section manager for the Embedded Systems and Communications Group; Allalaghata Pavan, Lee Graba, and Saul Cooper are research scientists in the Communications and Systems Architecture Department; and, Vipin Gopal, Sanjay Parthasarathy, and Saad Bedros are research scientists in the Systems and Control Technology Department, all at Honeywell Laboratories, Minneapolis, Minn. Contact them at {rakesh.jha, allalaghata.pavan, lee.graba, saul.cooper, vipin.a.gopal, sanjay.parthasarathy, saad.j.bedros}@honeywell.com.

Ionut Cardei is a former research scientist and Mihaela Cardei is a former intern in the Communications and Systems Architecture Department. Contact them at {ionut, mihaela}@cs.umn.edu.

Editors: Jerzy W. Rozenblit, University of Arizona, ECE 320E, Tucson, AZ 85721, jr@ece.arizona.edu; and Sanjaya Kumar, Motorola, 1501 W. Shure Dr., Arlington Heights, IL, 60004, sanjaya.kumar@motorola.com.

**Let your e-mail
address show
your professional
commitment.**



An IEEE Computer Society e-mail alias forwards e-mail to you, even if you change companies or ISPs.

you@computer.org

*The e-mail address
of computing professionals*



ICSE 2002



**May 19-25, 2002
Buenos Aires
Argentina**

Conference Website

<http://www.icse-conferences.org/2002/>

Call For Participation

<http://www.icse-conferences.org/2002/cfp.pdf>

Sponsoring Organizations:



Association for Computer Machinery



ACM Special Interest Group on Software Engineering



IEEE Computer Society
Technical Council on Software Engineering



Sociedad Argentina de Informática e Investigación Operativa