

Hierarchical Architecture for Real-Time Adaptive Resource Management

Ionut Cardei¹, Rakesh Jha², Mihaela Cardei¹, Allalaghatta Pavan²

Honeywell Technology Center

3660 Technology Drive, Minneapolis, MN 55418, USA

¹{ionut, mihaela}@cs.umn.edu, ²{jha, pavan}@htc.honeywell.com

Abstract. This paper presents the Real Time Adaptive Resource Management system (RTARM¹), developed at the Honeywell Technology Center. RTARM supports provision of integrated services for real-time distributed applications and offers management services for end-to-end QoS negotiation, QoS adaptation, real-time monitoring and hierarchical QoS feedback adaptation. In this paper, we focus on the hierarchical architecture of RTARM, its flexibility, internal mechanisms and protocols that enable management of resources for integrated services. The architecture extensibility is emphasized with the description of several service managers, including an object wrapper build around the NetEx real-time network resource management. We use practical experiments with a distributed Automatic Target Recognition application and a synthetic pipeline application to illustrate the impact of RTARM on the application behavior and to evaluate the system performance.

1 Introduction

Current distributed mission-critical environments employ heterogeneous resources that are shared by a host of diverse applications cooperating towards a common mission goal. These applications are generally a mix of hard-, soft- and non-real-time applications with different levels of criticality and have a variety of structures, ranging from periodic independent tasks, multimedia streams and parallel pipelines, to event-driven method-invocation communicating components. The applications usually tolerate a range of Quality of Services (QoS) and are ready to trade off QoS in favor of the most critical functions they perform. The distributed systems must be able to evolve and adapt to the high variability in resource demands and criticality of the applications as well as to the changing availability of resources.

The current industry trend is to build distributed environments for mission-critical applications using “Common-Off-The-Shelf” (COTS) commercial hardware and software components. A middleware layer above the COTS components provides consistent management for the system resources, decreases complexity and development costs.

¹ Funded by DARPA under NRAM Contract number N66001-97-C-8524.

This paper presents the Real Time Adaptive Resource Management system (RTARM), developed at the Honeywell Technology Center, that implements a general middleware architecture/framework for adaptive management for integrated services aimed to real-time mission-critical distributed applications.

The RTARM system has the following basic features [5]: (1) scalable end-to-end criticality-based QoS contract negotiation that allows distributed applications to share common resources while maximizing their utilization and execution quality; (2) end-to-end QoS adaptation that dynamically adjusts application resource utilization according to their availability while optimizing application QoS; (3) integrated services for CPU and network resources with end-to-end QoS guarantees; (4) real-time application QoS monitoring for integrated services and (5) plug-and-play architecture components for easy extensibility for new services.

The resource management architecture for RTARM uses an innovative approach that unifies heterogeneous resources and their management functions into a hierarchical uniform abstract service model [5]. The building block of the architecture is the Service Manager (SM). It encapsulates a set of services and their management functions and exports a common interface to clients and other service managers. This facilitates recursive hierarchies, in which heterogeneous services are integrated bottom-up. A higher-level service manager aggregates services provided by itself and its lower-level SMs and provides clients with a higher-level QoS representation.

In this paper, we focus on the architecture, protocols and implementation of an RTARM prototype that supports integrated services for real-time distributed applications. It runs as a middleware on a network of workstations and uses CORBA for portable communication. A major contribution of our work is the hierarchical feedback adaptation mechanism [1] that provides efficient dynamic QoS control for distributed data-flow applications. We illustrate the RTARM capabilities with a practical experiment with an Automatic Target Recognition (ATR) distributed application [9] and with a synthetic pipeline demonstration application.

The DARPA Quorum program [11] provides an extensive framework for QoS-related research projects. Similar efforts for building adaptive management systems for heterogeneous resources are GRMS [6,7], ARA [9,12], and QualMan [10]. GRMS is a precursor of RTARM. It introduced the uniform resource model and the atomic ripple scheduling protocol. Its hierarchical architecture reflects the application data flow and does not offer feedback adaptation. ARA considers a discrete set of runtime configurations for distributed applications and does feedback adaptation by resource reallocation. The ARA architecture is non-recursive and differs considerably from the uniform RTARM architecture by using proxies for specific service providers. QualMan is designed for multimedia applications and defines two basic resource management components, the resource scheduler and the QoS broker, that adhere to a uniform resource model without considering deeper recursive structures and QoS composition. [2] introduces a portable and QoS-enabled middleware platform suitable for building multimedia and real-time distributed applications.

The rest of this paper is organized as follows. Section 2 describes the RTARM hierarchical architecture, system models and interfaces. Section 3 presents the architecture of a Service Manager and describes the CPU, network and a higher-level SM. Section 4 continues with experiments involving an ATR application and

synthetic pipeline applications that emphasize the RTARM capabilities. The paper concludes in Section 5 with a discussion and future plans.

2 The RTARM System Architecture

We have designed and implemented the RTARM system prototype as a middleware layer above the operating system and network resources. The middleware approach provides the benefit of flexibility and portability but the increased distance to the basic resources makes fine-grained control difficult. The RTARM servers, developed in C++, run as user-level processes on Windows NT workstations and export a CORBA (Orbix [8]) interface to clients and applications. The RTARM model differentiates between clients and applications. A client is any entity that issues a request for services and negotiates a QoS contract that defines the allocated services. An application consumes services reserved by a client on its behalf and continuously cooperates with the resource management system to achieve the best available QoS while maintaining its runtime parameters within the contracted region. The QoS contract may change during the application lifetime.

2.1 The Service Manager Hierarchy

The RTARM system employs a hierarchical resource management architecture that facilitates provision of integrated services over heterogeneous resources. The uniform resource model [5] defines a recursive structural entity called Service Manager (SM) that encapsulates a set of resources and their management mechanism. At the bottom of the hierarchy are SMs that provide management functions for basic resources, such as CPU or network resources, and directly control resource utilization by application components. Higher level services are assembled on top of lower-level services, giving rise to a service hierarchy.

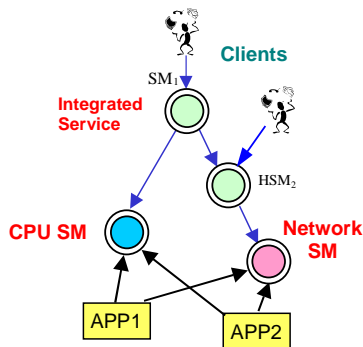


Fig. 1. Sample RTARM hierarchy consisting of one network SM, one CPU SM and two integrated service managers

Resources as well as negotiation requests are treated uniformly across the entire hierarchy. Higher-level service managers (HSM) may act as clients for lower-level SMs (LSM). The hierarchy allows dynamic configuration as new service managers can join the system at any time. A request for an integrated service sent to an HSM may require resources from lower-level service providers. The admission protocol builds a virtual reservation tree over the SM hierarchy that remains valid for the entire application lifetime. The SM hierarchy forms a directed acyclic graph, with SM as nodes and edges represented by the “uses-services-from” relation.

Figure 1 illustrates a simple RTARM hierarchy with two LSMs, a CPU and a Network SM, at the bottom of the hierarchy. Two clients request services from the two HSMs while applications are consuming CPU and network resources. Section 3 describes the service managers in more detail.

There are several benefits from a hierarchical, recursive, resource management architecture. First, services with complex, composite QoS representations are easier to implement on top of basic services. Complex distributed applications benefit from a richer representation of QoS. It simplifies the application design and facilitates consistent resource management for QoS-incompatible applications. Regardless of how complex the application architecture and QoS semantics are at the top of the SM hierarchy, at the bottom of the hierarchy everything translates to QoS requests for basic services (CPU and network in our prototype).

The hierarchical architecture of RTARM scales well with large distributed environments. Many SMs grouped in clusters may benefit from service localization and avoid communication bottlenecks. Sharing of LSMs between HSMs adds redundancy, fault tolerance and load balancing. In contrast, the centralized approach for heterogeneous resource management in distributed environments may introduce the drawbacks of a central controller: communication and processing bottleneck, one point of failure and decreased flexibility, but has certain performance benefits and lower latency.

A potential shortcoming for deep RTARM hierarchies derives from the increased distance between the top-most-level SM and bottom layer in the hierarchy. This may cause high latency for time sensitive RTARM functions, such as feedback adaptation and application control in case of deep SM hierarchies.

Issues related to deadlock prevention and distributed SM synchronization have been studied for the GRMS project [6,7] and can be easily extended to the RTARM model.

2.2 RTARM System Models

2.2.1 QoS Model and Translation. The quality of the interaction of a mission-critical application with a dynamic environment directly reflects its performance. The wide magnitude of this interaction requires a range for the quality measures. RTARM supports a multidimensional QoS representation, each dimension specifying an acceptable range $[Q_{\min}, Q_{\max}]$ of a quality parameter for the application. A set of range specifications, one per dimension, defines a QoS region. This QoS model facilitates resource negotiation and makes resource management more flexible.

In the RTARM recursive hierarchy, the QoS representation at a SM reflects the type of services provided by that SM. An HSM translates a QoS request for integrated services into individual QoS requests for services provided by itself and its lower-level SMs. When the SM receives replies from its LSMs, it reassembles the returned QoS into its own QoS representation in a process called QoS reverse-translation. RTARM uses a unique implementation for QoS, which is independent of the addressed service. We define a QoS parameter as a set of name-value pairs, where the value part is a sequence of one or more scalar primitive data values (string, short, double, etc.) and the name indicates the specific QoS dimension, such as “rate”, “workload”, “latency”, etc..

2.2.2 Adaptation Model. RTARM recognizes three situations when application QoS may be changed after admission [5]: (1a) QoS shrinking/reduction of lower criticality applications when a new application comes; (1b) QoS expansion/improvement when applications depart and release resources, and (2) feedback adaptation. While (1a) and (1b) imply contract changes and involve other applications, feedback adaptation does not change the contract but only varies the current operational point of the application within the contracted QoS region. Feedback adaptation is like closed loop control. It relies on monitoring of delivered QoS and uses the difference between delivered and desired QoS to adapt the application behavior.

2.3 RTARM Interfaces

Each SM implements and exports three interfaces: (1) *Negotiator* for admission control, collateral adaptation, QoS expansion and application control, such as suspend, resume and end; (2) *Service Manager* for SM hierarchy set up (register/deregister SM) and (3) *Monitor* for application monitoring and event propagation.

For admission control and adaptation RTARM uses a modified version of the GRMS Ripple Scheduling algorithm [6,7]. A detailed description with examples follows in Section 3.4. Briefly said, RTARM admission and adaptation employ a transaction-based two-phase commit protocol applied recursively at each SM. The first phase executes a service availability test starting from the SM that received the admission request, down on the reservation tree that resulted from the QoS translation and request dispatch process. The available, reserved QoS propagates back to the initiator SM from the lowest SM layer, being reverse-translated along the way. In the second phase, the initiator SM assesses the success status of the reservation phase and the transaction is committed or aborted, implying service reservations along the spanning tree to be committed, or to be cancelled, respectively. If not enough resources are available, a SM will try to adapt lower criticality applications to their minimum contracted QoS and use the released resources for the new application. Later, when resources become available, the SM expands the QoS for the most critical applications.

Sometimes in order to admit a new, more critical application, it is enough to squeeze the QoS of only a part of an existing distributed application. Then, changes in the high-level QoS may require collateral adaptation of other components of the

application that do not directly impact admission of the new application. For instance, for a multimedia stream application having frame rate as QoS parameter, if one processing stage is adapted to the minimum rate, than all other stages will run at the same low rate, too.

The next section presents the object architecture of the SM and details the implementation of a CPU, a Network and a Higher-level SM.

3 RTARM Service Managers

3.1 The Service Manager Architecture and Implementation

The unified resource model provides the benefits of a uniform internal architecture for all service managers (Figure 2) and a common interface between them.

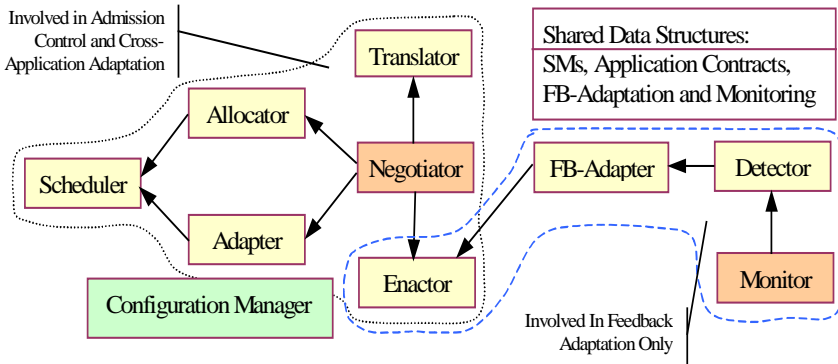


Fig. 2. The internal object architecture of a service manager

The arrows in the figure indicate object service requests. The components in a SM are as follows:

- *Negotiator*: brokers contract admission, delegates responsibilities to other components and exports the external RTARM CORBA interface.
- *Translator*: translates higher-layer integrated QoS into lower-layer QoS representation.
- *Allocator*: handles resource allocation/release when no adaptation is necessary.
- *Adapter*: handles resource allocation/release with adaptation and QoS expansion/contraction.
- *Scheduler*: determines whether allocation of resources and expansion of application QoS are feasible.
- *Enactor*: enforces changes in application QoS or status.

- *Monitor*: keeps an eye on applications in execution and passes status information and QoS usage to the Detector. Exports external RTARM CORBA interface.
- *Detector*: uses application runtime information (e.g. current QoS operational point) to detect significant changes in application operation (e.g. overload, underutilization, contract violation). Triggers Feedback Adapter actions.
- *Feedback Adapter*: decides corrective actions for applications when their runtime status changes significantly.

Additional data structures exist to hold information regarding application contracts, other service managers and available services.

Applications implement a simple CORBA interface that allows SMs to change their QoS and status. LSMs keep proxies for the application CORBA server objects. All RTARM CORBA servers and applications are started in the shared, multi-client activation mode.

A SM component class has the same object interface regardless of the SM position in the hierarchy or the resources the SM controls. For instance, the Adapter object implements the same functions in all SMs, but in a way that depends actually on the scope of the SM. Not all components are required within a SM. For example, a Translator may exist only inside an HSM.

RTARM provides a common object oriented execution framework that allows users to dynamically load SM components from shared libraries during runtime configuration. A configuration manager uses a mechanism similar to a *Factory Method* [4] to instantiate SM components. It also passes configuration information extracted from a configuration file to the SM components during their initialization.

For all SMs there is a single executable program that originally contains the empty SM framework and the configuration manager. By loading specialized components from shared libraries, the configuration manager practically starts different SMs. We use this technique when we initialize the CPU, Network and Higher-level SMs with components from specific Windows NT DLLs.

The flexibility of this plug-and-play feature permits implementation of a new SM by just replacing a set of components that realize a particular SM component interface, without rewriting the whole program. Writing a new SM component only requires the header file with the object interface, the executable program (common execution framework) and its corresponding library.

3.2 The CPU Service Manager

The CPU SM provides periodic applications access to a processor resource. Each computing node has a CPU SM, allowing concurrent applications to share a CPU. The application QoS is bi-dimensional: application execution rate (R) and iteration execution time (W) (Figure 3). The COP (Current Operational Point) represents the current values for the multidimensional QoS.

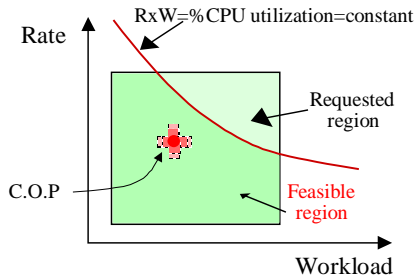


Fig. 3. CPU Service Manager QoS representation

3.2.1 Admission and Adaptation. The specific CPU scheduling policy is isolated within the Scheduler object and the Monitor keeps track of application CPU utilization. The invariant condition for admission and schedulability for n applications is $\sum_{i=1..n} R_i W_i < 100\%$ processor utilization. A more sophisticated CPU SM can be implemented at any time, by just using the plug-and-play feature, replacing the default Scheduler component with one specific to the scheduling discipline used.

The CPU SM service allocation unit for each periodic application is the fraction of CPU utilization ($R \times W$). The CPU SM communicates this information to applications and assumes they are well behaved and keep their process utilization below the allocated limits. The SM scheduler only assigns application rates and does not control the underlying OS scheduler. This policy works fine on a larger time scale and for our experimental purposes. For real-time performance one solution is to implement a soft real-time CPU scheduling server above the OS scheduler [10]. Commercial operating systems with soft real-time capabilities, like Windows NT and Solaris, limit the scheduler granularity to 10-20ms.

The CPU SM implements the Ripple Scheduling admission protocol. Because it is at the bottom of the SM hierarchy and has no LSMs, it does not make any other recursive calls. Adaptation and collateral adaptation reduce the application rate to the minimum contracted value. QoS expansion increases the application contracted QoS (rate) to the best available value.

3.2.2 Feedback Adaptation. The CPU SM controls the task rate in real-time. It cannot change the workload, which is left exclusively under application control. Applications send their current QoS operational point as events to the CPU SM monitor at the end of each periodic iteration. At any moment, the QoS COP may vary so that $R \times W \leq L$, where L is the fraction of the contracted processor utilization. The CPU SM adjusts the COP as follows: (1) increase rate when workload decreases; (2) decrease rate on overload, when the workload pushes the COP outside the contracted region.

3.3 The Network Service Manager

We integrated the NetEx real-time network management system [3,13] from Texas A&M University into the RTARM system. NetEx runs as middleware and provides connection-oriented real-time communication with guaranteed delay and bandwidth over COTS network infrastructure, such as ATM and switched 10/100 Mbps Ethernet. NetEx uses a tri-dimensional QoS: period, delay and message size and adds the connection source and destination network addresses to the connection contract. The NetEx resource management interface is, however, incompatible with the RTARM interfaces. It has different semantics and it does not export the two-phase commit protocol.

We built an object-oriented *wrapper* [4] around NetEx that hides the incompatibilities and exports the RTARM interface to clients, applications and HSMs (Figure 4). The wrapper method can be used to integrate any service provider in the RTARM architecture.

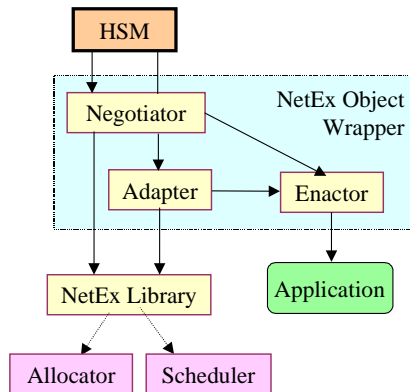


Fig. 4. The object wrapper for NetEx communication manager

The wrapper implements three SM components, Negotiator, Adapter and Enactor, that map the RTARM interface calls for admission, adaptation and expansion to the native NetEx API. NetEx does not provide feedback adaptation for connections, so the wrapper SM does not implement feedback adaptation either. It is important to note, however, that our HSM for integrated services for parallel pipeline applications implements hierarchical feedback adaptation. This is detailed in the next section.

3.4 The Higher-Level Service Manager for Integrated Services

Within the RTARM service manager hierarchy, HSMs aggregate services from LSMs (CPU, Network or any other type of SM) and provide RTARM services to

applications that need a more complex QoS representation. The unified resource model enables recursive deployment of HSMs. Our HSM implementation is generic and is able to support various types of distributed applications with arbitrary QoS representations that map to available LSM QoS. The only restriction is that the Ripple Scheduling admission and adaptation procedure and the hierarchical feedback adaptation must not contradict the application semantics. The QoS Translator SM component inside an HSM is responsible for translating a QoS request into something the LSMs understand. Replacing the translator component with a different one (for a different QoS representation) produces a HSM capable of supporting different integrated services.

3.4.1 Admission and Adaptation. The Negotiator implements the recursive two-phase admission protocol that runs at the heart of each HSM. The code for the first phase, reservation, follows next:

```

test_reservation(in reqQoS, out avQoS, in candidates,
                out adaptedApps)
// reqQoS is the requested QoS region
// avQoS is the returned (acceptable) QoS region
// candidates is the list of applications that may be
// adapted in order to accommodate the current request
// adaptedApps is the list of adapted applications
{
  translate reqQoS into:
    LS - list of requested services from LSMs, and
    LreqQoS - corresponding QoS per service.
  for each service S from LS {
    for each LSM lsm that provides service S {
      success = lsm->test_reservation(LreqQoS[lsm],
                                     lsmAvQoS[S],
                                     candidates that run on lsm,
                                     lsmAdaptedApps[S])
      if success then mark admitted service
        and continue with next service S from LS
    }
    if service S was not admitted then {
      cancel all previous successful reservations
      return false
    }
  }
  // now all services from LS have been admitted
  reverse-translate and maximize the returned QoS from
    lsmAvQoS into avQoS
  perform collateral adaptation if necessary
  return true
}

```

The second phase that commits the resource reservation from phase I is implemented like this:

