

Dynamic Scheduling and Resource Management in Heterogeneous Computing Environments with Reconfigurable Hardware

Steven P. Smith

QuickFlex, Inc.
Austin, Texas

Abstract – Dynamically reconfigurable hardware resources within complex heterogeneous computing environments show tremendous potential for an improvement in system performance, a reduction in the necessary number of processing resources, an increase in system degradability in the presence of faults, and an expansion of system flexibility, field upgradeability, and system lifecycles. However, efficient management and task scheduling for these resources demands the consideration of a number of factors beyond those necessary for more traditional systems. The document addresses the key issues associated with the exploitation of dynamically reconfigurable hardware. The patented Dynamic Logic Run-Time Resource Manager (DLRMTM), a middleware architecture developed specifically to address these challenges, is presented.

Keywords: Heterogeneous, reconfigurable hardware, resource management, scheduling, FPGA.

1 Introduction

Task scheduling and resource management in heterogeneous computing environments has been the subject of research efforts for more than two decades^{2,11,12}. And while truly optimal scheduling within a complex, heterogeneous computing environment according to a relevant set of criteria remains intractable, practical and effective strategies have emerged for distributed computing environments composed of various general-purpose, software-programmable processing resources.

However, the addition of run-time reconfigurable hardware resources to such heterogeneous computing environments expands the scheduling and resource management challenges considerably. Further complicating matters is the growing feasibility of dynamic partial reconfiguration of FPGA resources. The hardware multi-tasking that such capabilities make possible greatly increases the number of choices for task assignment available to the scheduler and raises the question of how best to select hardware behaviors for replacement when necessary. Launching tasks on reconfigurable hardware requires a number of additional steps not relevant to traditional software-programmable computing resources

with fixed hardware, notably including loading configuration bit streams, an operation that may take longer than the task execution itself.

Complex heterogeneous computing environments with reconfigurable hardware are increasingly likely to contain FPGA resources from different device vendors and subsystem developers, potentially using entirely different access mechanisms and programming interfaces. Some tasks may be capable of executing on multiple different reconfigurable subsystems, while others may be inescapably tied to a single resource.

Coarse-grained, heterogeneous computing environments with run-time reconfigurable hardware elements are particularly well suited for scientific computing, complex real-time instrumentation and control systems (e.g., avionics), communications, and related applications. While some of these application domains simply seek to make efficient use of the resources available for a given workload, many of them entail mission-critical real-time task processing in which missed deadlines can have disastrous consequences.

The remainder of this document presents a dynamic scheduling and resource management middleware architecture by QuickFlex, Inc. The Dynamic Logic Run-Time Resource Manager (DLRMTM) system presents an application-level interface to software elements that neatly abstracts the implementation of a specific task behavior from its interface, enabling the underlying functionality to be mapped onto a number of different computing resources within the system. By design, the DLRM architecture is completely independent of FPGA device vendor or type.

The DLRM builds on the core concepts of the Common Object Request Broker Architecture (CORBA)³, the real-time extensions to CORBA contained in RT-CORBA⁹, and the Open ORB Framework. The next sections present a brief overview of the relevant CORBA, RT-CORBA, and Open ORB¹ concepts and a short discussion on scheduling strategies. Following the overview is a discussion of the complicating factors inherent in the use of dynamically reconfigurable hardware in a distributed computing environment as well as the requirements imposed by the objective of exploiting partial dynamic

reconfiguration. The QuickFlex DLRM architecture is then presented, followed by a summary of the DLRM's status, future development plans, and conclusions.

It is important to note that while the DLRM described later has borrowed heavily from the concepts in CORBA, it also contains some enhancements to and deviations from the basic CORBA design intended to maximize its performance and efficiency in its target deployment environments. These differences will be highlighted as they are presented.

2 Basic Concepts of the Common Object Request Broker Architecture

The Common Object Request Broker Architecture was developed by the Object Management Group (OMG) to separate cleanly the implementation of objects from their interfaces and invocation. In so doing, CORBA enables applications to make use of heterogeneous computing resources transparently. Note that CORBA defines interfaces and not implementations. These interfaces are specified using the CORBA Interface Definition Language (IDL)³.

The primary objective of the CORBA system is to automate a number of aspects central to the development of distributed applications, including: object location, object connection management, memory management, parameter marshalling and de-marshalling, event and request sequencing, error handling, object activation, and concurrency, among others.

CORBA uses a client-server model, adding an object request broker (ORB) between the two to effect the clean separation of the client from the server. A high level view of CORBA processing is shown in Figure 1.

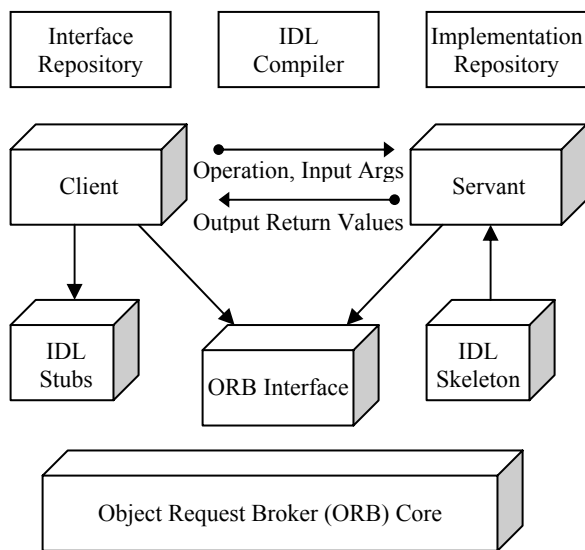


Figure 1
CORBA Middleware Architecture

Key concepts of the CORBA model include *object references*, which are strongly typed and opaque handles to objects, *objects*, which contain an identity, and interface, and an implementation, *clients*, which make requests on objects using an object reference, *servers*, which represent the process context for objects, *stubs*, which convert object method invocations into messages, *skeletons*, which convert messages back into method calls, *servants*, which implement requests on one or more objects, and the core *object request broker*, which provides the basic message passing infrastructure.

Processing in CORBA occurs via object methods according to the following steps:

- 1) The client locates the target object,
- 2) The server is activated, providing the execution context for the object implementation (servant),
- 3) The client sends a request message to the server,
- 4) The server activates the object implementation (servant),
- 5) The client waits for the request to complete,
- 6) The servant processes the request,
- 7) The server returns the result data to the client,
- 8) And the client is unblocked and has control returned to it, resuming its execution.

In addition to providing a small flavor of the operating environment embodied by CORBA, the sequence above will be used in subsequent sections to contrast the extra processing requirements of real-time processing and of dynamically reconfigurable hardware.

3 Real-time CORBA

Real-time CORBA⁹ extends the basic CORBA architecture to encompass quality-of-service (QoS) control. Policies and mechanisms in RT-CORBA enabling these extensions fall into three categories: processor resources, communication resources, and memory resources. Processor resources include thread pools, priority models, and platform-independent priorities. Communication resources include protocol resources and explicit binding. Memory resources include request buffering.

Thread pools are provided by servers for the use of object servants. Portable priorities are an extension to the core object request broker, while scheduling services and end-to-end propagation of priorities are situated between the client and servant. Explicit binding and protocol properties are also extensions to the ORB, as are standard mechanisms for synchronization.

It is useful here to consider some basic concepts of scheduling in distributed systems. Single-level scheduling can be divided into three cases. The first single-level case

is when scheduling occurs entirely independently on each processing resource within the system. There is no end-to-end propagation of priorities in this case. This approach is typical of scheduling strategies in non-real-time distributed systems.

In the second single-level scheduling case, each node is still responsible for its own scheduling independent of other nodes, but an application that spans nodes propagates its end-to-end priority and timeliness requirements. Each node in the system then uses this input to schedule the task in competition with all other tasks enqueued on that node. This case leads to coherent scheduling that is not globally optimized.

In the third single-level scheduling case, there is a globally cooperative scheduling strategy implemented on each node, with communication among the nodes in order to ensure that a globally optimum dynamic schedule is maintained. This approach is intractable in general, but it is included in the discussion for the sake of completeness.

Finally, there is multi-level scheduling, which combines the first two single level scheduling cases above in an effort to adapt to dynamic system conditions and to seek a more globally optimal solution. RT-CORBA does not explicitly support multi-level scheduling, nor does it preclude it. The architecture is shown in Figure 2.

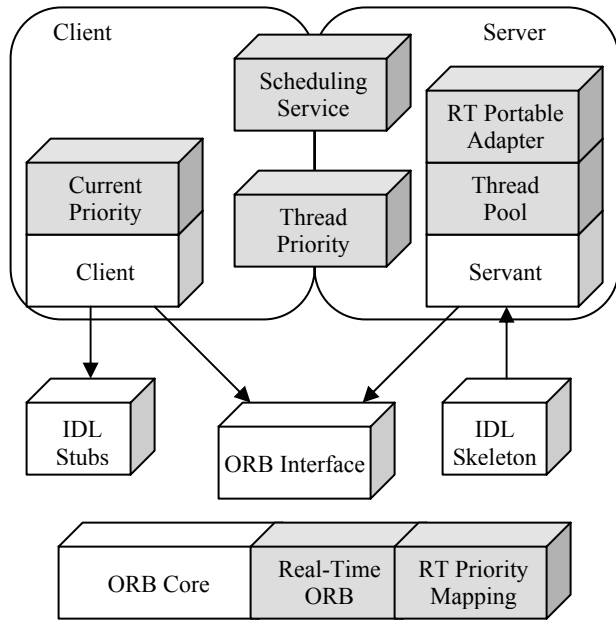


Figure 2
RT-CORBA Middleware Architecture

RT-CORBA defines the notion of a *distributable thread* as its fundamental abstraction for application execution. A distributable thread may span multiple nodes in the system, but it still represents a single logical thread of control in the invoking application. RT-CORBA applications are typically made up of a number of

distributable threads. Distributable threads contain one or more *scheduling segments*, which are regions of execution that have their own scheduling parameters. The shaded elements in Figure 2 summarize the additions to Figure 1 that are a part of RT-CORBA.

4 Overview of Scheduling Strategies

A very large number of scheduling strategies have been reported in the literature, but for the purposes at hand, it is enough to discuss only a basic handful of them. The simplest approach to scheduling is to employ fixed priorities. In systems with periodic task schedules and fixed execution times, it is possible to develop a static schedule based on *rate monotonic analysis* (RMA), which assigns the highest priority to the task that has the greatest frequency. RMA scheduling can ensure that critical schedules are met, but at the cost of inefficient resource utilization and higher system hardware costs. However, the ability to perform a priori scheduling using RMA greatly reduces the necessary complexity of the run-time scheduling support.

The *earliest deadline first* (EDF) scheduling strategy assigns the highest priority to the thread with the soonest deadline. EDF can be applied statically when the thread deadlines and execution times are known and fixed, but it can also be applied dynamically. So called “soft real-time” systems often use dynamic EDF scheduling strategies.

The *least laxity first* (LLF) scheduling strategy dynamically selects threads for execution based on their current laxity, which is the thread’s deadline minus the sum of the current time and the estimated time remaining for the execution of the thread. On soft real-time systems, the LLF scheduling strategy seeks to minimize the tardiness of task completions.

The *maximized accrued utility* (MAU) scheduling strategy uses a utility function that may be thread-specific to determine when to execute threads. The MAU scheduler seeks only to maximize the value of the utility functions over a period of task executions. Note that the MAU strategy is remarkably flexible, enabling individual distributable threads to set their own criteria for scheduling.

Finally, note that while RT-CORBA envisions task priorities will be set either by the server or propagated from the client, it also supports the notion of a priority transform, which is a user-supplied function that can be used to implement other strategies for setting priorities. Related to this is the concept of resource managers in RT-CORBA, which can exploit the priority transform mechanism to dynamically modify the priority of a distributable thread based on various relevant criteria, such as resource loading or input channel congestion.

A core weakness of RT-CORBA is the fact that resource-specific characteristics and requirements in a heterogeneous system can only be reflected in the scheduling process through the priority transform mechanism. As a result, this information is essentially external to the RT-CORBA standard.

5 The Open ORB Framework

The Open ORB Framework¹ defines powerful task and resource models that directly address the shortcomings of the base RT-CORBA specification. Resources can be modeled at varying levels of abstraction, and are closely related to the task models. Each task is associated with a pool of resources. Resources are defined hierarchically, with pools of resources coming together to define top-level *virtual task machines* that specify the set of resources needed for a given application. Such virtual task machines typically span a number of resource types, such as memories, threads, and processing elements

Objects in the Open ORB Framework can invoke other objects associated with different tasks. These events are referred to as *task switching points*, and present an opportunity to change the resource pool available to a given task both dynamically and adaptively.

The concepts of hierarchical resource pools and task switching points in Open ORB Framework represent a powerful way to think about one-to-many object mappings in which multiple resources in a system can execute a given task. Such object bindings which can be mapped to multiple computing resources can be modified in a task-specific manner at task switching points. Changes in the virtual task machine (the dynamic resource pool associated with a given task) may be based on factors such as current resource loading, communication cost, channel congestion, and the availability of alternative task resources.

Figure 3 shows an example of a virtual task machine for a sample task.

6 Scheduling and Resource Management in the Dynamic Logic Run-time Resource Manager

The DLRM makes use of RT-CORBA's priority transform mechanism and the virtual task machine concept from the Open ORB Framework to effect scheduling and resource management for heterogeneous computing environments with reconfigurable hardware resources. At any given time, a task can be mapped onto any of a number of hardware resources and resource types. For example, a specific decompression task may have implementations (servants) available for one or more different FPGA resource types in the system, a digital

signal processor, and various general-purpose CPUs in the computing environment.

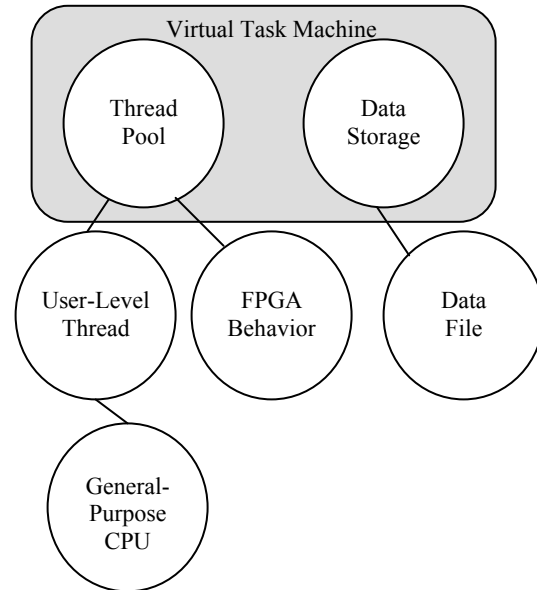


Figure 3
Example Virtual Task Machine in Open ORB

6.1 Selecting an Object Servant Host in Systems with Reconfigurable Computing Resources

When selecting among candidate general-purpose host computing resources for a given object servant, it is generally sufficient to consider only the estimated execution time of the object invocation on the platform, the platform's current loading, and perhaps the estimated communication costs associated with the message and parameter passing.

Reconfigurable computing resources, however, present a number of further complications. Where the selection of which local memory resources on general-purpose processors to unload to make room for the new servant is well taken care of by the local scheduler and the basic functioning of the CPU's cache replacement policy, the selection of which hardware behaviors to replace on a currently "full" reconfigurable computing resource is less well defined. Considering only the current priority of the candidate behaviors overlooks the cost of the initial load of the behavior. Instead, a separate utility function is used to drive the evaluation of candidate hardware behaviors to be replaced to make room for a new object servant.

The mapping of the task within an application to a specific computing resource may change dynamically over the course of application execution, potentially at each task switching point. In making the determination of whether to change the binding of an object reference from one resource to another, the DLRM resource manager

employs a utility function with weighted input parameters that may include:

- the estimated time cost of remapping the object binding,
- the penalty measure associated with any displacement of object servants the new mapping may require (based on the displaced object’s priority, execution frequency, and estimated load time),
- the estimated time cost of moving any data required by the object,
- the congestion of any communication channels required by the proposed mapping,
- the current congestion level of the channels used by the current mapping,
- the estimated execution time of the object servant on both the current and candidate resources,
- and the loading of the current and candidate resources.

Since some of the tangible computing resources in a system may not support the execution of the general-purpose software needed to implement the scheduling services and resource management functions, computing resources in the DLRM are grouped into a special case of a virtual task machine hierarchy that contains specific computing resources at the level of the leaf node, with the next higher layer of the hierarchy containing general-purpose computing resources currently responsible for implementing the CORBA and RT-CORBA scheduling and resource management functions for the leaf node computing resources under the general-purpose resource in the hierarchy.

Higher layers of the resource management hierarchy are used to group the first-line resource managers together into larger clusters of resources that are “close” together. This special resource management hierarchy is designed to be modifiable at run-time if individual resource managers determine that they lack sufficient resources for a given application or applications, or if they determine that scheduling overhead is too great relative to average object execution times. In the latter case, the computing resources may be redistributed to two or more general-purpose computing resources capable of executing the scheduling and object request broker functions.

6.2 Standardizing the DLRM Management Interface to Reconfigurable Hardware Resources

The Dynamic Logic Run-time Resource Manager employs a multi-level interface to individual reconfigurable hardware resources. As with CORBA at the application level, this interface separates the DLRM

access mechanism, which is specific to each type of FPGA resource and the subsystem that hosts it, from the core behavior of the DLRM. The first level of this interface is the Reconfigurable Resource Driver, which provides a standard application-programming interface to the DLRM software that enables basic data read and write access to the FPGA resource.

The hardware behavior that realizes the other end of the subsystem-specific resource driver, the DLRM support port on each FPGA-based subsystem, is loaded into the device during power-on reset and remains resident throughout system operation. The implementation of this device-level support behavior may be of any type that provides the required access. It is necessary, however, that the access port be continuously available to the management software. Access to the port may be protected through encrypted data and commands.

The DLRM support port on each reconfigurable computing resource provides internal device access to the individual hardware behavior containers currently hosted by the system. These three elements provide full and standardized access to the FPGA resource by the DLRM software. The elements of the DLRM support access chain are shown in Figure 4.

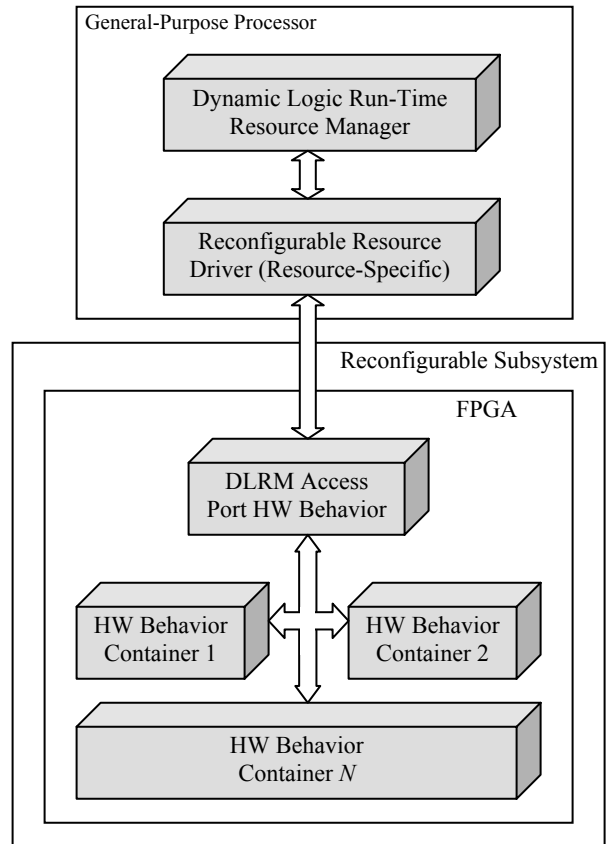


Figure 4
Standardized DLRM Device Access Chain

6.3 Management and Characteristics of Reconfigurable Hardware Behavior Alternatives

In heterogeneous computing systems with multiple reconfigurable hardware computing resources, it may be possible to map a given hardware object behavior into different FPGA resources or into different positions within a single FPGA. Compilation and generation of the bit stream variants is beyond the scope of the Dynamic Logic Run-time Resource Manager, but the various alternative configuration bit streams that realize a given behavior are all managed by the DLRM and are used to drive the resource selection process upon hardware object invocation.

In the case of devices supporting partial reconfiguration, it is necessary for individual hardware behaviors to be mapped into devices on the FPGA fabric that support the partial reconfiguration process and its limitations with respect to behavior boundaries⁶. The DLRM ensures that hardware behaviors managed by the middleware meet these requirements by placing a container behavior around all independent hardware behaviors at the time they are compiled and mapped into specific devices within the FPGA. This is the same behavior container that provides support access to the DLRM.

As mentioned above, beyond ensuring compliance with partial reconfiguration mapping restrictions, the DLRM behavior containers also provide a standardized maintenance and management interface for each behavior currently loaded into the reconfigurable resource. In addition to the behavior containers, each FPGA is required by the DLRM to present a standard access mechanism and interface for status queries, logging of execution statistics (run-time, execution frequency, time of last invocation, etc.). The standardization of these supporting hardware behaviors enables the clean separation of the DLRM management functions from the implementation details of the reconfigurable hardware resources in a system.

The device-level DLRM support behavior must provide access to the following information to the DLRM software:

- device type and speed grade,
- device-level DLRM access behavior type and version,
- current number and types of behavior containers hosted by the FPGA,
- timer for execution times, with DLRM read access,
- various control functions, including timer stop, start, and reset,

- read and write access to any optional device or application-specific registers and information that may be present,
- and DLRM behavior container monitor port and data path characteristics (e.g., bit width).

Similarly, each behavior container within the reconfigurable computing resource must provide the following information to the DLRM via the monitor access port:

- behavior container type (i.e., size) and version,
- name and version of the hardware behavior currently loaded in the container,
- identity of the behavior's current owning task,
- and read and write access to any optional registers or behaviors within the container.

Some additional information that may also be provided by the behavior containers includes:

- time of most recent invocation of the behavior hosted by the container,
- elapsed time of the most recent invocation,
- and a moving average of elapsed execution times for the currently hosted behavior.

6.4 DLRM Task Scheduling

The DLRM exploits the flexibility and extensibility of the *maximized accrued utility* (MAU) scheduling strategy, which supports the inclusion of the many unusual characteristics of dynamically reconfigurable hardware relevant to task scheduling decisions. While the utility function used for MAU scheduling of reconfigurable resources could become quite complex due to the number of factors that affect the optimal schedule for a given task load, the DLRM is currently working with simplified utility functions that seek primarily to account for the load-time of target behavior and the estimated execution time of the task on that reconfigurable subsystem host.

The default utility function for MAU scheduling in the DLRM is of the form

$$F = C_d \cdot d \left(\frac{C_l}{t_l} + \frac{C_e}{t_e} \right) \quad (\text{Eq. 1})$$

where C_d is a weight for the resource distance term, d is a measure of the distance from the client, C_l and C_e are weights for the load and execution times, and t_l and t_e are the load and execution times themselves. The DLRM also supports user-supplied utility functions.

7 DLRM Status

The Dynamic Logic Run-time Resource Manager version 2.0 is currently in testing on the new QuickFlex reconfigurable computing platform supporting the Xilinx Virtex-4 device family with a high pin-count footprint and the resource manager executing on an AMDAlchemy general-purpose processor based on the MIPS architecture and running Linux. The fixed processor controls access to the Virtex-4 device and enables the stand-alone operation of the reconfigurable platform if desired. The Virtex-4 is connected so that all of the device configuration access options are available for the Virtex-4. The parallel configuration port also serves as the DLRM support and maintenance port for the FPGA. Both the Alchemy processor and the Virtex-4 device have access to ample dedicated dynamic memory.

In addition to using multiple instances of our hardware platform connected via local networking, QuickFlex also expects to augment its development and test platform with commercially available reconfigurable resources hosting other reconfigurable devices.

One shortcoming of the CORBA concept when applied to reconfigurable hardware that has already emerged is the need to provide a short-cut path from applications and clients to the servants and hardware resources that will fulfill their processing demands without having to incur the entire burden of the CORBA access sequence illustrated in Figure 1. To enable high performance access, QuickFlex is designing the notion of *session-based* scheduling and resource management. The intent of this work is to allow applications to incur minimal overhead in the invocation of each task call by establishing a session that will serve as a sort of extended task. The clear challenge in making this concept operationally successful will be the trade-off between minimizing the overhead of each task call while not forcing otherwise inefficient scheduling and resource allocation to occur.

8 Conclusions

Reconfigurable computing resources in heterogeneous distributed systems present some unique challenges for task scheduling and resource management, especially in a real-time environment. The Dynamic Logic Run-time Resource Manager architecture effectively addresses these challenges while still supporting other computing resources such as digital signal processors and general-purpose CPUs. Based on the concepts of RT-CORBA, the DLRM enables high system performance and efficient utilization of system resources while maintaining full device vendor independence. The DLRM also preserves all of the software implementation-independent characteristics of the CORBA architecture and adds hierarchical resource management of the form embodied

in the Open ORB Framework. Version 2.0 of the DLRM is currently in production testing and will soon have its general release.

9 References

- [1] G.S. Blair, et al., "The Design and Implementation of Open ORB Version 2," *IEEE Distributed Systems Online*, vol. 2, no. 6, 2001.
- [2] Raymond K. Clark, *Scheduling Dependent Real-Time Activities*, Computer Science Doctoral Dissertation, Carnegie Mellon University, August, 1990.
- [3] "Common Object Request Broker Architecture: Core Specification, Version 3.0.3, Object Management Group, March, 2004.
- [4] Hector A. Duran-Limon, Gordon S. Blair, and Geoff Coulson, "Adaptive Resource Management in Middleware: A Survey," *IEEE Distributed Systems Online*, Volume 5, No. 7, July, 2004.
- [5] Richard F. Freund, Michael Gherrity, et al, "Scheduling Resources in Multi-User, Heterogeneous, Computing Environments with SmartNet," *Proceedings of the 7th IEEE Heterogeneous Computing Workshop*, March 1998, pp. 184-199.
- [6] Mark Goosman, Nij Dorairaj, and Eric Shiflet, "How to Take Advantage of Partial Reconfiguration in FPGA Designs," *Programmable Logic Design Line*, February 6, 2006.
- [7] V. Kalogeraki, P.M. Melliar-Smith, and L.E. Moser, "Dynamic Scheduling for Soft Real-Time Distributed Object Systems," *Proceedings of the 3rd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, IEEE CS Press, 2000, pp. 114-121.
- [8] David L. Levine, Christopher D. Gill, and Douglas C. Schmidt, "Dynamic Scheduling Strategies for Avionics Mission Computing," *Proceedings of the 17th IEEE/AIAA Digital Avionics Systems Conference*, November, 1998.
- [9] "Real-Time CORBA Specification, Version 2.0," Object Management Group, November, 2003.
- [10] Jon Siegel, "CORBA for Real-Time Systems," *Java Developer's Journal*, October, 2000.
- [11] John A. Stankovic, "Distributed Real-Time Computing: The Next Generation," *Journal of the Society of Instrument and Control Engineers of Japan*, January, 1992.
- [12] Steve Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments," *IEEE Communications Magazine*, February, 1997.