A Framework for (Re)Deploying Components in Distributed Real-time and Embedded Systems

N. Shankaran[†], J. Balasubramanian[†], D. Schmidt[†], G. Biswas[†]
P. Lardieri[‡], E. Mulholland[‡], and T. Damiano[‡]
[†]Dept. of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN
[‡]Lockheed Martin Advanced Technology Labs, Cherry Hill, NJ

Abstract

This paper describes the Resource Allocation and Control Engine (RACE) that integrates multiple resource management algorithms for (re)deploying and managing performance of application components in distributed real-time and embedded (DRE) systems. RACE enables DRE systems to (re)configure allocation and control algorithms depending on application characteristics and environmental conditions. It also enables developers to focus on algorithm logic, while reusing many mechanisms used to (re)configure and (re)deploy the algorithms on distributed computing nodes.

1. INTRODUCTION

Component-based technologies are increasingly being applied to distributed real-time and embedded (DRE) systems, such as shipboard computing environments, avionics mission computing systems, and earth science missions. Applications in DRE systems have a range of quality of service (QoS) requirements that may vary in response to changes in mission goals at runtime, *e.g.*, due to new information or because certain tasks cannot be completed on time. QoS requirements can also vary due to changes in system runtime performance, *e.g.*, due to loss of resources, transient overload and/or algorithmic properties.

To support different types of applications running in various DRE system environments, a range of resource management *algorithms* and *mechanisms* are needed to (1) allocate resources to application components and (2) control system QoS and behavior after application components have been deployed. Allocation algorithms determine the initial deployment process by mapping application components to the appropriate target nodes; control algorithms adapt the execution of application components at runtime in response to changing environments and variations in resource availability and/or demand. Mechanisms perform the allocation and control decisions of these algorithms.

A common way to develop resource management for DRE systems is to handcraft algorithms and mechanisms for specific application use cases. Although this approach can yield efficient *point solutions*, it can also lead to a number of problems. First, the *aggregate* QoS of multiple applications may be suboptimal if resource management algorithms are tightly coupled to a subset of the system's requirements and operating conditions that change dynamically. Second, handcrafted point solutions can increase system complexity due to the following tedious and error-prone human-intensive programming tasks: (1) specifying and evaluating component resource requirements, (2) examining application behavioral and interaction characteristics to identify which resource management algorithm(s) are best suited for (re)deployment, (3) monitoring resources available in the system to assist allocation and adaptation decisions, and (4) interacting with middleware infrastructure mechanisms that (re)configure and (re)deploy components based on decisions made by allocation and control algorithms.

This paper describes the structure, functionality, and use of the *Resource Allocation and Control Engine* (RACE), which is an open-source framework (`www.dre.vanderbilt.edu`) that addresses the problems described above using standard OMG Lightweight CORBA Component Model (CCM) middleware (`www.omg.org`). RACE helps manage various system resources (such as network bandwidth, as well as CPU and memory on each node) by selectively applying algorithms designed to meet application QoS requirements and operating conditions. RACE also separates resource allocation and control algorithms from the underlying middleware mechanisms so that these algorithms can reuse common mechanisms to (re)configure and (re)deploy components properly onto DRE system nodes.

2. OVERVIEW OF RACE

RACE is an extensible framework that factors out mechanisms common to different allocation and control algorithms, including (1) capabilities for describing application QoS characteristics declaratively, (2) monitors that track application and infrastructure performance and resource usage, (3) the ability to represent allocation/control algorithm policies declaratively and automatically configure the resource management middleware to enforce these policies, and (4) the ability to (re)deploy and (re)configure the application components based on decisions made by the allocation and control algorithms. Applications in DRE systems can use RACE to manage system resource utilization and ensure their QoS requirements are met even under varying operational contexts and/or resource requirement/availability.

Figure 1 illustrates the architecture of the RACE framework, which is implemented as an assembly of CCM components in CIAO and DAnCE [2]. Each RACE component is

SAC '06, April, 23-27, 2006, Dijon, France.

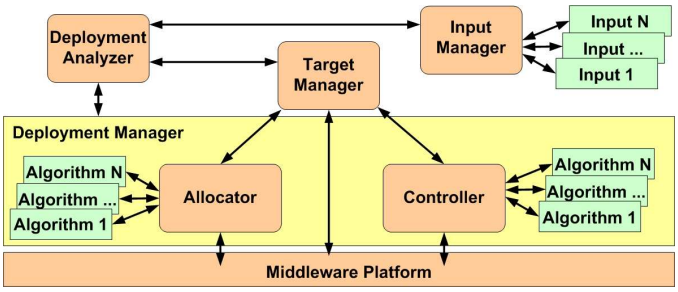

Figure 1: The RACE Architecture

described below and is motivated in terms of the problems described in Section 1.

Specifying and evaluating component resource requirements. Application characteristics and QoS requirements determine the resource management algorithms to use when making (re)deployment decisions. RACE's `InputManager` is a CCM component that interacts with external users and planners to capture key properties of system (re)deployment, including which application components comprise the system, the execution profiles and resource requirements of each component, the data/control dependencies between components, the communication characteristics of components, and the QoS requirements of those communications in terms of latency, throughput, etc. The `InputManager` receives this input from various sources, including component modeling languages, such as PICML (`www.dre.vanderbilt.edu/cosmic`) and declarative languages, such as web service interfaces. To reconcile different formats from various input sources, RACE's `InputManager` captures the input in the form of standards-based XML descriptors defined by the OMG D&C specification.

Supporting multiple resource management algorithms. Applications in DRE systems often have different QoS goals and requirements, so a single allocation and control algorithm may not meet the needs of these applications. RACE's `DeploymentManager` is a CCM component that supports the (re)configuration and operation of multiple resource management algorithms, whose behavior depends on application characteristics and environmental conditions. It contains an `Allocator` that supports multiple resource allocation algorithms, such as PBFD [1], and Avala [5] whose behavior depends on the requirements captured by the `InputManager`. It also contains a `Controller` that supports multiple control algorithms, such as FCS [4] and HySUCON [3], that help ensure QoS properties of components, as specified to the `InputManager`. RACE's `DeploymentManager` component captures the allocation and control decisions of the algorithms in the form of standards-based XML descriptors defined by the D&C specification and passes these descriptors to the middleware platform, which then performs system re(deployment).

Monitoring dynamic system information. To make accurate allocation and adaptation decisions, system resource utilization must be monitored periodically. RACE's `TargetManager` is a CCM component that monitors online system resource utilization, so that allocation/adaptation decisions and QoS properties of applications can be evaluated dynamically. The `TargetManager` monitors the (potentially heterogeneous) network of nodes where application components are deployed based on the decisions made by `Allocator` algorithms and periodically informs `Controller` algorithms to aid resource management adaptation decisions.

Support for selecting proper resource management algorithms. Configuring all resource management algorithms for a particular DRE system can yield excessive memory footprint and resource usage. Application behavioral characteristics and requirements must therefore be analyzed carefully to (1) operate the best possible algorithms and (2) change the algorithms if the QoS becomes unsatisfactory. RACE's `DeploymentAnalyzer` is a CCM component that parses the behavioral characteristics and requirements of applications and identifies the best resource management algorithms to use for the system (re)deployment.

3. APPLICATION OF RACE

We are applying RACE to several large-scale DRE systems, including Naval shipboard computing systems and NASA earth science missions. Our goal is to evaluate how well it monitors system QoS and adapts algorithms in response to changing application requirements and operating conditions. We are also evaluating how well different resource management algorithms optimize the deployment of application components.

To show RACE in action, we briefly describe its application to Naval shipboard computing systems. In this example, the `DeploymentAnalyzer` examines the input from the `InputManager` and if there are no communications across components, it uses a classical simple bin-packing algorithm to allocate components to nodes. Conversely, if component communications are involved, RACE uses a more sophisticated bin-packing algorithm, such as PBFD [1].

RACE's `DeploymentAnalyzer` can (re)configure the algorithms to use by analyzing (1) the system properties and deployment constraints captured by the `InputManager` and (2) the system monitoring information from `TargetManager` to ensure the QoS properties of the system are maintained. After components are mapped to nodes using the appropriate `Allocator` algorithms, RACE's `Controller` algorithms adapt application component execution dynamically in response to changing requirements or variations in resource availability or demand. For example, a `Controller` can (1) modify an application's current operating mode, (2) dynamically update component implementations, and/or (3) redeploy components to other target nodes to meet end-to-end QoS requirements.

4. REFERENCES

[1] D. de Niz and R. Rajkumar. Partitioning Bin-Packing Algorithms for Distributed Real-Time Systems. *International Journal of Embedded Systems*, 2005.

[2] G. Deng, J. Balasubramanian, W. Otte, D. C. Schmidt, and A. Gokhale. DAnCE: A QoS-enabled Component Deployment and Conguration Engine. In *Proceedings of the 3rd Working Conference on Component Deployment*, Grenoble, France, Nov. 2005.

[3] X. Koutsoukos, R. Tekumalla, B. Natarajan, and C. Lu. Hybrid Supervisory Control of Real-Time Systems. In *11th IEEE Real-Time and Embedded Technology and Applications Symposium*, San Francisco, California, Mar. 2005.

[4] C. Lu. *Feedback Control Real-Time Scheduling*. PhD thesis, University of Virginia, Charlottesville, VA, May 2001.

[5] M. Mikic-Rakic, S. Malek, and N. Medvidovic. Improving Availability in Large, Distributed Component-Based Systems Via Redeployment. In *3rd International Working Conference on Component Deployment (CD 2005)*, Grenoble, France, 2005.