ADAPTIVE RESOURCE MANAGEMENT ALGORITHMS, ARCHITECTURES, AND FRAMEWORKS FOR DISTRIBUTED REAL-TIME EMBEDDED SYSTEMS

By

Nishanth Shankaran

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

December, 2008

Nashville, Tennessee

To Amma and Appa for their love, support, motivation, and guidance

To Sujata for all the encouragement

ACKNOWLEDGMENTS

I am grateful to the following individuals for their guidance, support, and encouragement during my graduate education at the University of California, Irvine and Vanderbilt University. Without their help, this dissertation would not have been possible.

I would like to begin by thanking Prof. Douglas C. Schmidt, my advisor and mentor, for giving me an opportunity for working with him at the Distributed Object Computing (DOC) group at Vanderbilt University. During my graduate education at Vanderbilt University, Doug has been my biggest source of encouragement, guidance, support, and inspiration, and I am ever grateful to him for that. I have learned a lot from Doug on a wide genre of topics, not limited to just academic ones. Although my role as one of his advisees is coming to an end, I hope my role as one of his protégés would continue in the future. Next, I would like to thank my co-advisors, Dr. Xenofon D. Koutsoukos, Prof. Gautam Biswas, and Dr. Chenyang Lu, for their guidance and collaboration over the past four years. I am thankful to Prof. Janos Sztipanovits for serving on my dissertation committee and Dr. Raymond Klefstad for giving me an opportunity to work at the DOC group at University of California, Irvine.

My graduate education has been supported by various organizations and agencies. I would like to take this opportunity to thank the following people: Richard Schantz and Joe Loyall at BBN Technologies, Boston, for providing us with the UAV scenario used in this dissertation; Patrick Lardieri, Ed Mulholland, and Tom Damiano at Lockheed Martin Advanced Technology Labs, Cherry Hill, for providing the initial motivation for RACE; Dipa Suri and Adam S. Howell at Lockheed Martin Advanced Technology Center, Palo Alto, for collaborating with us on numerous projects, and providing us with many interesting and challenging scenarios such as the MMS mission system and the SEAMONSTER sensor-web used as case studies in this dissertation.

Over the past six years, I have learned a lot from both past and present DOC group members, both at Vanderbilt University and University of California, Irvine. I would like to thank Mark Panahi and Krishna Raman at the DOC group at UC Irvine for bringing me up to speed on CORBA and ZEN. I would like to thank Krishnakumar Balasubramanian, John S. Kinnebrew, Arvind Krishna, Jeff Parsons, and Nilabja Roy at the DOC group at Vanderbilt University for many fruitful discussions and brain-storming sessions over the years.

I am thankful to the following people for making my stay at Irvine enjoyable: Yogita Bhasin, Nirupama Srinivasan, Vinay Chandrasekhar, Narayan Vishvanathan, Srinivas Kollu, and Ashish Bhargave. When I moved from Irvine to Nashville in the summer of 2004, Jaiganesh Balasubramanian eased my transition and helped me to settle down; I'm grateful to him for that. Ever since I moved to Nashville, the time I spent with the following people were always enjoyable and memorable: Krishnakumar Balasubramanian, Abhishek Dubey, Amogh Kavimandan, Arvind Krishna, Manish Kushwaha, and Anantha Narayanan.

Last but not the least, I would like to thank my family: my parents for their love, support, motivation, and guidance through all these years; my fiancé, Sujata, for all the encouragement. But for their support and encouragement, this dissertation would have been nothing but a figment of my imagination.

Nishanth Shankaran

Vanderbilt University

24thOctober 2008

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Distributed real-time and embedded (DRE) systems form the core of many mission-critical domains, such as shipboard computing environments [68], avionics mission computing [72], multi-satellite missions [78], and intelligence, surveillance and reconnaissance missions [71]. *Quality of service (QoS)-enabled distributed object computing (DOC) middleware* based on standards like Real-time Common Object Request Broker Architecture (RT-CORBA) [62] and the Real-Time Specification for Java (RTSJ) [10] have been used to develop such DRE systems. More recently, *QoS-enabled component middleware*, such as the Lightweight CORBA Component Model (CCM) [57] and PRiSm [73], have been used to build such systems [72]. As middleware technologies are being used extensively to develop such complex systems, a summary of the evolution of middleware technology is presented next.

I.1 Evolution of Middleware Technology

This section summarizes the evolution of various middleware technologies used to build DRE systems, primarily focusing on their contributions and limitations.

I.1.1 Distributed Object Computing (DOC) Middleware

Commercial-off-the-shelf (COTS) middleware technologies for DOC based on standards such as The Object Management Group (OMG)'s CORBA [58] and Sun's Java RMI [89], encapsulates and enhances native OS mechanisms to create reusable network programming components. These technologies provide a layer of abstraction that shields application developers from the low-level platform-specific details and define higher-level distributed programming models whose reusable APIs and components automate and extend native OS capabilities. Figure 1 shows the architectural layout of a DOC application built atop CORBA based middleware.

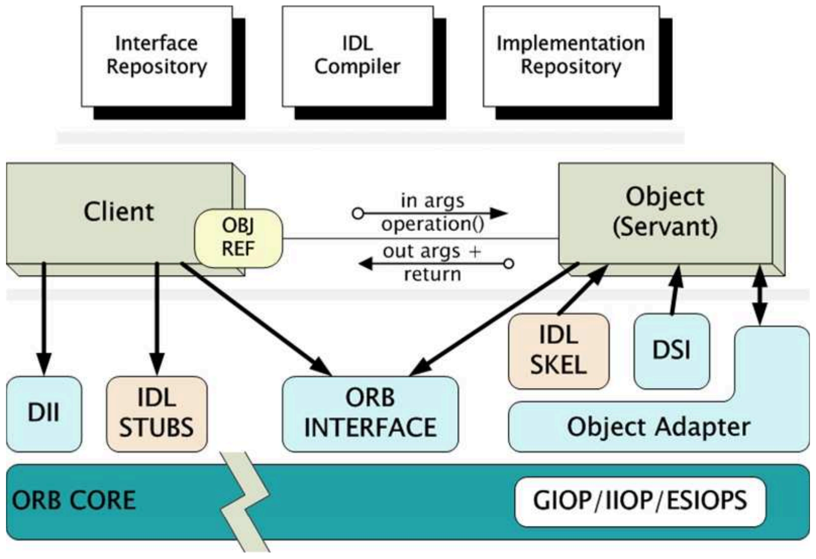


Figure 1: Model of an Application Built atop CORBA Middleware

Conventional DOC middleware technologies, however, address only *functional* aspects of system/application development such as how to define and integrate object interfaces and implementations. They do not address QoS aspects of system/application development such as how to (1) define and enforce application timing requirements, (2) allocate resources to applications, and (3) configure OS and network QoS policies such as priorities for application processes and/or threads. As a result, the code that configures and manages QoS aspects often become entangled with the application code.

I.1.2 QoS-enabled DOC Middleware

Limitations of conventional DOC middleware identified above have been addressed by middleware standards such as RT-CORBA [62] and RTSJ [10]. As shown in Figure 2, middleware based on these technologies support explicit configuration of QoS middleware aspects such as priority and threading models, provide many real-time features including end-to-end priority propagation, scheduling service, and explicit binding of network connections.

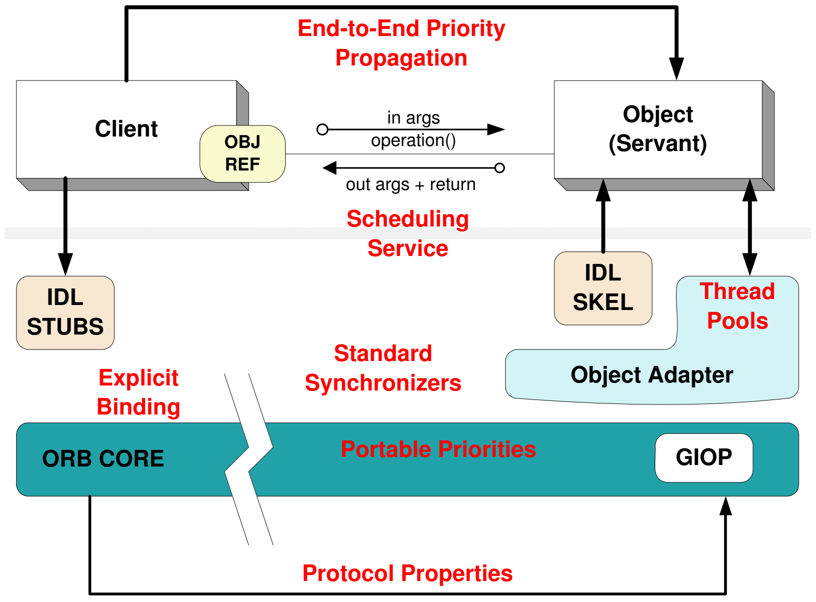


Figure 2: Model of an Application Built atop RT-CORBA Middleware

However, these technologies do not provide a higher level abstraction that separates real-time policy configuration from the application functionality. Thus, they lacks support for system design and development of large-scale systems. QoS-enabled DOC middleware support only the design and development of individual application objects. They lack generic standards for (1) distributing object implementations within the system, (2) installing, initializing, and configuring objects, and (3) interconnection between independent objects, all of which are crucial in development of a large-scale DRE system. Therefore, when large-scale distributed systems are built using QoS-enabled DOC middleware technologies, system design and development is tedious, error prone, hard to maintain and/or evolve, and results in a brittle system.

I.1.3 Conventional Component Middleware

Component middleware technologies, such as the CORBA Component Model (CCM) [60] and Enterprise Java Beans [5, 77] provide capabilities that addresses the limitation of DOC middleware technologies in the context of system design and development. Examples of additional capabilities offered by conventional component middleware compared to conventional DOC middleware technology include (1) standardized interfaces for application component interaction, (2) model-based tools for deploying and interconnecting components, and (3) standards-based mechanisms for installing, initializing, and configuring application components, thus separating concerns of application development, configuration, and deployment.

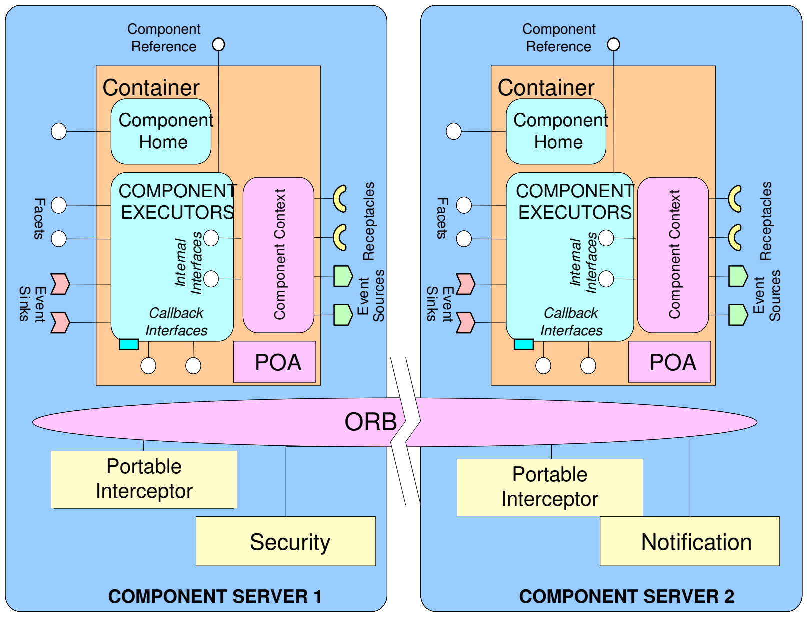


Figure 3: Entities of a CORBA Component Middleware

CCM is built atop CORBA object model, and therefore, system implementors are not tied to any particular language or platform for their component implementations. As shown in Figure 3, key entities of CCM-based component middleware include:

- **Component,** which encapsulates the behavior of the application. Components interact with clients and each other via *ports*, which are of four types: (1) *facets*, also known as provided interfaces, which are end-points that implement CORBA interfaces and accept incoming method invocations, (2) *receptacles*, also known as required connection points, that indicate the dependencies on end-points provided by another component(s), (3) *event sources*, which are event producers that emit events of a specified type to one or more interested event consumers, and (4) *event sinks*, which are event consumers and into which events of a specified type are pushed. The programming artifact(s) that provides the "business logic" of the component is called an *executor*.

- **Container,** which provides an execution environment for components with common operating requirements. The container also provides an abstraction of the underlying middleware and enables the component to communicate via the underlying middleware bus and reuse common services offered by the underlying middleware.

- **Component Home,** which is a factory [32] that creates and manages the life cycle for instances of a specified component type.

- **Component Implementation Framework (CIF),** which defines the programming model for defining and constructing component implementations using the Component Implementation Definition Language (CIDL). CIF automates the implementation of many component features which include generation of programming skeletons and association of components with component executors with their context and homes.

- **Component Server,** which is a generic server process that hosts application containers. One or more components can be collocated in one component server.

Component middleware provides a standard "virtual boundary" around application components, defines standard container mechanisms needed to execute components in generic component servers, and specifies a reusable/standard infrastructure needed to configure and deploy components throughout a distributed system. Although conventional component middleware support design and development of large scale distributed systems, they do not address the address the QoS limitations of DOC middleware. Therefore, conventional component middleware can support large scale enterprise distributed systems, but not DRE systems that have the stringent QoS requirements.

I.1.4 QoS-enabled Component Middleware

To address the limitations with various middleware technologies listed above, QoS-enabled component middleware based on standards such as the OMG Lightweight CCM [57] and Deployment and Configuration (D&C) [61] specifications have evolved. One such middleware is the Component Integrated ACE ORB (CIAO) [81], which combines the capabilities of conventional component middleware and QoS-enabled DOC middleware.

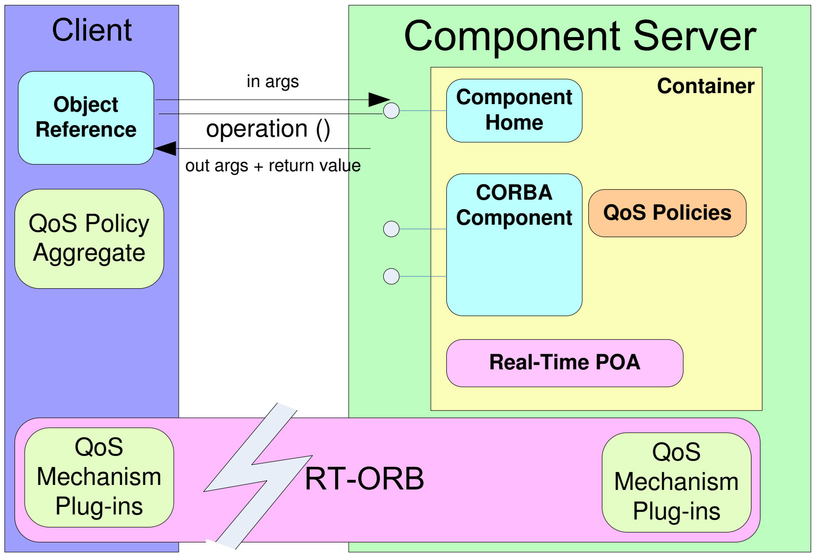


Figure 4: Entities of QoS-Enabled CORBA Component Middleware

As shown in Figure 4, QoS-enabled component middlewares offer explicit configuration of QoS middleware parameters that affect the real-time performance of the system.

Since QoS-enabled component middleware technologies are built atop conventional component middleware technologies, QoS-enabled component middlewares inherit the capabilities that aid in design and development of large scale distributed systems from conventional component middlewares. In summary, QoS-enabled component middleware capabilities enhance the design, development, evolution, and maintenance of DRE systems [80].

I.2 Overview of Research Challenges

Middleware technologies provide capabilities that address some, but by no means all, important DRE system development challenges. Some of the remaining key challenges in developing, deploying, configuring, and managing large-scale DRE systems using middleware technologies include:

Runtime Management of *Multiple* System Resources. Many mission-critical DRE systems execute in *open* environments where system operational conditions, input workload, and resource availability cannot be characterized accurately *a priori*. Achieving high end-to-end quality of service (QoS) is an important and challenging issue for these types of systems due to their unique characteristics, including (1) constraints in multiple resources (*e.g.*, limited computing power and network bandwidth) and (2) highly fluctuating resource availability and input workload. Conventional resource management approaches, such as rate monotonic scheduling [42], are designed to manage system resources and providing QoS in *closed* environments where operating conditions, input workloads, and resource availability are known in advance. Since these approaches are insufficient for open DRE systems, there is an increasing need to introduce resource management mechanisms that can *adapt* to dynamic changes in resource availability and requirements.

A promising solution is *feedback control scheduling* (FCS) [2, 22], which employs software feedback loops that dynamically control resource allocation in response to changes in input workload and resource availability. These techniques enable adaptive resource management capabilities in DRE systems that can compensate for fluctuations in resource availability and changes in application resource requirements at run-time. When FCS techniques are designed and modeled using rigorous control-theoretic techniques and implemented using QoS-enabled software platforms, they can provide robust and analytically sound QoS assurance.

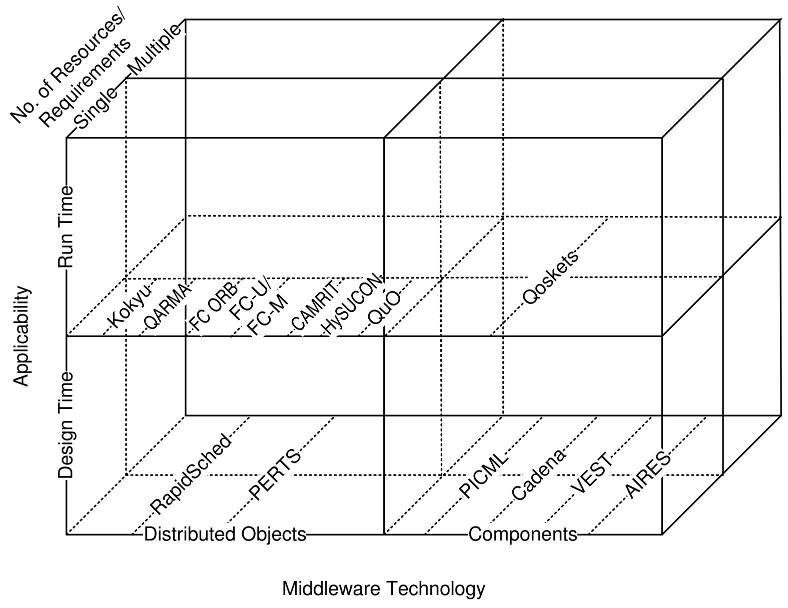


Figure 5: Taxonomy of Related Research

As shown in Figure 5, although existing research have been shown to be effective in managing a single type of resource, they have not managed multiple types of resources. It is still an open issue, therefore, to extend individual resource management algorithms to work together to manage multiple types of resources in a *coordinated* way, such as managing computational power and network bandwidth simultaneously.

Complexity of Deploying and Configuring Resource Management Algorithms in DRE Systems. In the past, significant research has been done in designing and developing general purpose, as well as domain specific, resource management algorithms for dynamic systems. Example of general purpose resource management algorithms include EUCON [52], HySUCON [41], and FC-U/FC-M [51]. Examples of domain/use-case specific resource management algorithms include CAMRIT [82] and HiDRA [70].

Since domain specific resource management algorithms are (usually) built for a specific use-case/domain, such resource management algorithms and mechanisms may be effective for that domain; however, they cannot be easily reused in another domain. On the other hand, in order for general purpose resource management algorithms to be used in real world systems in a portable way, they have to be implemented either in the middleware or the OS. However, this requires solid understanding of both the middleware/OS and the algorithm, which is hard. Moreover, the cost of employing another algorithm might be high since it might involve reimplementation of significant portions of the middleware/OS.

As a result, many resource management algorithms have been developed based on strong theoretical foundations; however, only a few algorithms "see the light of day", *i.e.*, evaluated in real systems. Therefore, what is missing is a easily customizable resource management framework that reuses entities of a resource management mechanism – monitors, resource management algorithm(s), and effectors – across domains in a portable manner and enables "plug & play" of new/domain-specific entities.

I.3 Research Approach

To address the challenges identified in Section I.2, this dissertation presents a detailed overview of (1) adaptive resource algorithms and architectures to manage multiple resource in DRE systems and (2) a fully configurable middleware based adaptive resource management framework. A brief summary of the different aspects of this dissertation is presented below.

1. **Hierarchical Distributed Resource-management Architecture**

   To address the challenges identified in Section I.2 in the context of adaptive management of multiple system resources, this dissertation presents a control-based multi-resource management architecture – Hierarchical Distributed Resource-management

Architecture (HiDRA). HiDRA employs a control-theoretic approach featuring two types of feedback controllers that coordinate the utilization of computational power and network bandwidth to prevent over-utilization of system resources. This capability is important because processor overload can cause system failure, and network saturation can cause congestion and severe packet loss. Subject to the constraints of the desired utilization, HiDRA improves system QoS by modifying appropriate application parameters. Chapter II describes HiDRA in detail.

2. **Resource Allocation and Control Engine**

   To address the challenges identified in Section I.2 in the context of deploying and configuring resource management algorithms in DRE systems, this dissertation presents the *Resource Allocation and Control Engine* (RACE), which is an adaptive resource management framework built atop CIAO. RACE provides reusable entities – resource monitors, application/system QoS monitors, resource allocators, controllers, and effectors – that can be reused across domains. Moreover, RACE can be configured with domain specific implementation of the above mentioned entities. Chapter III describes RACE in detail.

I.4 Research Contributions

Our research on adaptive resource management for DRE systems has resulted in algorithms and architectures that perform adaptive management of *multiple resources* at run-time and a fully configurable resource management framework that compliments theoretical research on adaptive resource management and enables the deployment and configuration of feedback control loops in DRE systems. The key research contributions of our work on HiDRA and RACE are shown in Table 1.

Category	**Benefits**
Adaptive Resource Management Algorithms and Architectures (HiDRA)	1. A novel algorithm and architecture for runtime management of multiple system resources using control theoretic techniques. 2. Provides a resource management architecture that ensures utilization of multiple resources converge to the specified set-point. 3. Improves system QoS.
Adaptive resource management framework (RACE)	1. A fully configurable adaptive resource management framework for DRE systems, 2. Enables the deployment and configuration of resource management feedback control loops in DRE systems, 3. Details three case-studies where RACE has been successfully applied.

Table 1: Summary Of Research Contributions

I.5 Dissertation Organization

The remainder of this dissertation is organized as follows: Chapter II focuses on adaptive resource management algorithms and architectures and describes the related research, the unresolved challenges, our research approach to solve these challenges, and empirical evaluation of our research on runtime management of multiple system resources. Chapter III focuses on adaptive resource management frameworks for DRE systems, describes the related research, the unresolved challenges, our resource management framework – RACE – and how RACE addresses these unresolved challenges, and an empirical evaluation of RACE. Chapters IV, V, and VI, focus on three DRE system case studies where RACE has been successfully applied and presents an overview of the resource management requirement of each system, description of how RACE addressed these requirements, and an empirical evaluation of the resource management capabilities of RACE in each of the case studies. Chapter VII presents concluding remarks, provides a summary of lessons learned from our research on adaptive resource management for DRE systems, and outlines future research.

CHAPTER II

ADAPTIVE RESOURCE MANAGEMENT ALGORITHMS AND ARCHITECTURES

As described in Chapter I, DRE systems form the core of many mission-critical domains, including autonomous air surveillance [71], total ship computing environments [68], and supervisory control and data acquisition systems [11, 17, 29]. Often, these systems execute in *open* environments where system operating conditions, input workload, and resource availability cannot be characterized accurately *a priori*. These characteristics are beginning to emerge in today's large-scale systems of systems [21], and they will dominate in the next-generation of ultra-large-scale DRE systems [38]. Achieving high end-to-end quality of service (QoS) is important and challenging for these types of systems due to their unique characteristics, including (1) constraints in multiple resources (*e.g.*, limited computing power and network bandwidth) and (2) highly fluctuating resource availability and input workload.

Conventional resource management approaches, such as rate monotonic scheduling [42, 45], are designed to manage system resources and providing QoS in *closed* environments where operating conditions, input workloads, and resource availability are known in advance. Since these approaches are insufficient for open DRE systems, there is a need to introduce resource management mechanisms that can *adapt* to dynamic changes in resource availability and requirements. A promising solution is *feedback control scheduling* (FCS) [2, 22, 50], which employs software feedback loops that dynamically control resource allocation to applications in response to changes in input workload and resource availability. These techniques enable adaptive resource management capabilities in DRE systems that can compensate for fluctuations in resource availability and changes in application resource requirements at runtime. When FCS techniques are designed and modeled using rigorous control-theoretic techniques and implemented using QoS-enabled software platforms, they can provide robust and analytically sound QoS assurance.

Although existing FCS algorithms have been shown to be effective in managing a single type of resource, they have not been enhanced to manage multiple types of resources. It is still an open issue, therefore, to extend individual FCS algorithms to work together in a *coordinated* way to manage multiple types of resources, such as managing computational power and network bandwidth simultaneously. To address this issue, we have developed a control-based multi-resource management framework called *Hierarchical Distributed Resource management Architecture* (HiDRA). HiDRA employs a control-theoretic approach featuring two types of feedback controllers that coordinate the utilization of computational power and network bandwidth to prevent over-utilization of system resources. This capability is important because processor overload can cause system failure and network saturation can cause congestion and severe packet loss. HiDRA improves system QoS by modifying appropriate application parameters, subject to the constraints of the desired utilization.

This dissertation provides contributions to both theoretical and experimental research on FCS. Its theoretical contribution is its use of control theory to formally prove the stability of HiDRA. Its experimental contribution is to evaluate empirically how HiDRA works for a real-time distributed target tracking application built atop *The ACE ORB* (TAO) [67], which is an implementation of Real-time CORBA [62]. Our experimental results validate our theoretical claims and show that HiDRA yields desired system resource utilization and high QoS despite fluctuations in resource availability and demand by efficient resource management and coordination for multiple types of resources.

The remainder of the chapter is organized as follows: Section II.1 describes the architecture and QoS requirements of our DRE system case study; Section II.2 compares our research on HiDRA with related work; Section II.4 explains the structure and functionality of HiDRA; Section II.5 formulates the resource management problem of our DRE system case study described in Section II.1 and presents an analysis of HiDRA; Section II.6 empirically evaluates the adaptive behavior of HiDRA for our DRE system case study; and Section II.7 concludes the chapter by presenting a summary.

II.1 Case Study: Target Tracking DRE System

This section describes a real-time distributed target tracking system that we use as a case study to investigate adaptive management of multiple system resources in a representative open DRE system. The tracking system provides emergency response and surveillance capabilities to help communities and relief agencies recover from major disasters, such as floods, hurricanes, and earthquakes. In this system, multiple unmanned air vehicles (UAVs) fly over a pre-designated area (known as an "area of interest") capturing live images. The architecture of this distributed target tracking system, which is similar to other reconnaissance mission systems [49] and target tracking systems [19, 20], is shown in Figure 6.

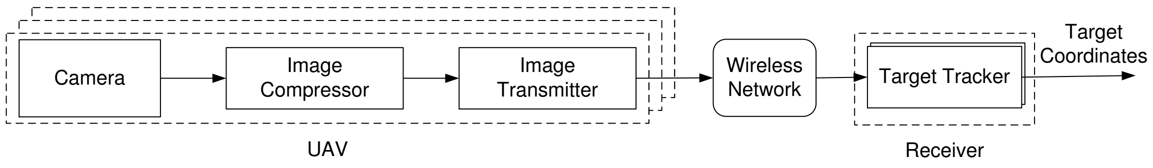


Figure 6: Target Tracking DRE System Architecture

Each UAV serves as a data source, captures live images, compresses them, and transmits them to a receiver over a wireless network. The receiver serves as a data sink, receives the images sent from the UAVs, and performs object detection. If the presence of an object of interest is detected in the received images, the tracking system determines the coordinates of the objects automatically and keeps tracking it. The coordinates of the object is reported to responders who use this information to determine the appropriate course of action, *e.g.*

initiate a rescue, airlift supplies, etc. Humans, animals, cars, boats, and aircraft are typical objects of interest in our tracking system.

The QoS of our resource-constrained DRE system is measured as follows:

- *Target-tracking precision*, which is the distance between the computed center of mass of an object and the actual center of mass of the object, and

- *End-to-end delay*, which is the time interval between image capture by the UAV and computation of the coordinates of an object of interest. End-to-end delay includes image processing delay at the UAV, network transmission delay, and processing delay of the object detection and tracking sub-system at the receiver.

Just as any real-time system, end-to-end delay is a crucial QoS in our emergency response system and must be as low as possible. A set of coordinates computed with a lower precision and lower end-to-end delay is preferred over a set of coordinates computed with a higher precision and/or higher end-to-end delay.

There are two primary types of resources that constrain the QoS of our DRE system: (1) *processors* that provide computational power available at the UAVs and the receiver and (2) the *wireless network bandwidth* that provides communication bandwidth between the UAVs and the receiver. To determine the coordinates accurately, images captured by the UAVs must be transmitted at a higher quality when an object is present. This in turn increases the network bandwidth consumption by the UAV. To increase the utility of the system, images are transmitted at a higher rate by the UAVs when objects of interest are present in the captured images. This in-turn increases the processor utilization at the receiver node, and thus increases the processing delay of the object detection and tracking sub-system. Moreover, transmission of images of higher quality at a higher rate increases the bandwidth consumption by the UAV. If the network bandwidth is over-utilized considerably, the network transmission delay increases, which in-turn increases the end-to-end delay.

Utilization of system resources (*i.e.*, wireless network bandwidth and computing power at the receiver) are therefore subject to abrupt changes caused by the presence of varying numbers of objects of interest. Moreover, the wireless network bandwidth available to transmit images from the UAVs to the receiver depends on the channel capacity of the wireless network, which in-turn depends on dynamic factors, such as the speed of the UAVs and the relative distance between UAVs and the receiver due to adaptive modulation [1, 36].

The coupling between the utilization of multiple resources, varying resource availability, and fluctuating input workloads motivate the need for adaptive management of multiple resources. To meet this need, the captured images in our system are compressed using JPEG, which supports flexible image quality [79]. Likewise, we choose to use image streams rather than video because video compression algorithms are computationally expensive, the computation power of the on-board processor on the UAVs is limited, and emergency response and surveillance applications and operators do not necessarily need video at 30 frames per sec. However, the computational power of the UAV on-board processor is large enough to compress images of the highest quality and resolution and transmit them to the receiver without overloading the processor.

In JPEG compression, a parameter called the *quality factor* is provided as a user-specified integer in the range 1 to 100. A lower quality factor results in smaller data size of the compressed image. The quality factor of the image compression algorithm can therefore be used as a *control knob* to manage the bandwidth utilization of an UAV. To manage the computational power of the receiver, end-to-end execution rate of applications is used as the control knob.

II.2 Related Research

Resource management algorithms and architectures have been studied extensively in the research community. As shown in Figure 7, this research can be broadly categorized into two categories based on their applicability: (1) design time solutions and (2) runtime solutions. These two categories are discussed in detail below.

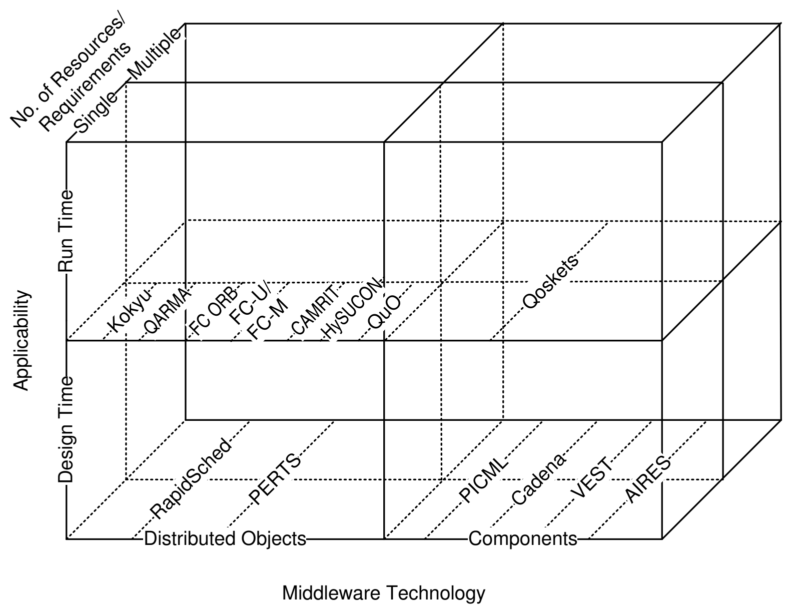


Figure 7: Taxonomy of Related Research

- **Design time solutions.**

  Design time resource management solutions have been historically studied under the context of scheduling algorithms and feasibility analysis. Classical scheduling algorithms include rate monotonic scheduling [42], fixed priority scheduling algorithm, deadline driven scheduling algorithm, and a mixed scheduling algorithm presented in [45]. These algorithms assume that the deadline of a periodic tasks is equal to it period. The work presented in [7] relaxes this assumption and presents a scheduling algorithms where the deadline of tasks can be less than their periods. The research presented in [63] describes a feasibility analysis for hard real-time periodic tasks. Classical bin-packing algorithms [46] can also be viewed as resource management algorithms since they can be used to allocate resource to applications. The research presented in [24] describes a heuristic based approach to solve the multi-dimension bin-packing problem.

- **Runtime solutions.**

  A number of control-theoretic approaches have been applied to DRE systems to overcome limitations with traditional scheduling approaches that are not suited to handle dynamic changes in resource availability and result in a rigidly scheduled system that adapts poorly to change. A survey of these techniques is presented in [2].

  Feedback control scheduling (FCS) [50] is designed to address the challenges of applications with stringent end-to-end QoS executing in open DRE systems. These algorithms provide robust and analytical performance assurances despite uncertainties in resource availability and/or demand. FC-U and FC-M [51] and HySUCON [41] employ control-theoretic techniques to manage the processor utilization on a single node. EUCON [52] presents a control-theoretic approach to manage processor utilization on multiple nodes simultaneously.

  A hierarchical control scheme that integrates resource reservation mechanisms [22, 44] with application specific QoS adaptation [12] is proposed in [3]. This control scheme features a two-tier hierarchical structure: (1) a global QoS manager that is responsible for allocating computational resources to various applications in the system and (2) application-specific QoS managers/adapters that modify application execution to use the allocated resources efficiently and improves application QoS.

  The middleware control framework is presented in [43] manages the performance of a distributed multimedia application. The objective of this framework is to ensure that global system wide properties, such fairness between competing applications, as well as QoS requirement of individual applications are met, without over utilizing system resources. This research utilizes task control model and fuzzy control model to enhance the QoS adaptation decision of multimedia DRE systems. However, the control framework established in this work is still confined to single type of resource, *(i.e.)*, transmission rate in a distributed visual tracking system.

CAMRIT [82] applies control-theoretic approaches to ensure transmission deadlines of images over an unpredictable network link and also presents analytic performance assurance that transmission deadlines are met.

II.3 Unresolved Challenges

Design time solutions are efficient at managing system resources and QoS in *closed* environments where operating conditions, input workloads, and resource availability are known in advance. These approaches, however, cannot be applied to DRE systems that execute in *open* environments where system operational conditions, input workload, and resource availability cannot be characterized accurately *a priori*.

As shown in Figure 7, existing runtime solutions perform resource management of only one type of system resource, *i.e.*, either computing power *or* network bandwidth. For DRE systems, these approaches are insufficient since multiple types of resources are to be managed simultaneously, and in a coordinated fashion. One approach to manage both computing power and network bandwidth might use either the hierarchical control structure proposed in [3], FC-U/FC-M, HySUCON, or EUCON to manage the processor utilization, and use CAMRIT to manage the network bandwidth utilization. Unfortunately, this approach does not take into consideration the coupling between the two types of system resources and does not necessarily guarantee system stability.

To address these challenges, this dissertation presents a detailed overview of the design and implementation of a distributed adaptive resource management architecture that yields predictable and high performance resource management and coordination for multiple types of system resources.

II.4 The Hierarchical Distributed Resource-management Architecture (HiDRA)

This section presents the *Hierarchical Distributed Resource-management Architecture* (HiDRA), which employs a control-theoretic approach to manage processors and network bandwidth simultaneously. Our control framework is shown in Figure 8 and consists of three entities: *monitors*, *controllers*, and *effectors*. A monitor is associated with a specific

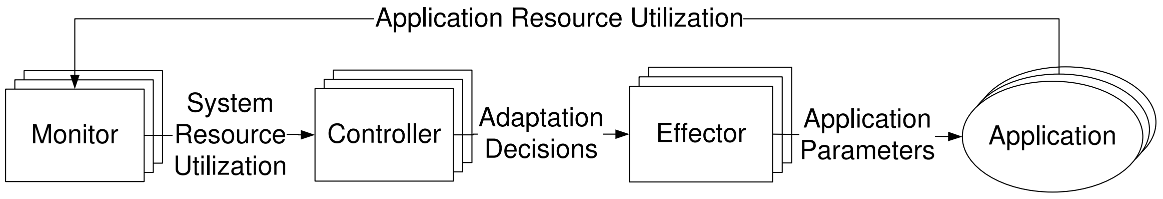


Figure 8: The HiDRA Control Framework

system resource and periodically updates the controller with the current resource utilization. The controller implements a particular control algorithm and computes the adaptations decisions for each application (or a set of applications) to achieve the desired system resource utilization. Each effector is associated with an application and modifies application parameters to achieve the controller-recommended application adaptation.

We proceed to instantiate the HiDRA control framework for the domain of target tracking described in Section II.1. Each application in our DRE system is composed of two subtasks: *image compression* and *target tracking*. To ensure end-to-end QoS, therefore, resource utilization of both subtasks must be controlled. As shown in Figure 9, HiDRA consists of two types of feedback control loops: (1) a processor control loop located at the receiver that manages the processor utilization and (2) a bandwidth control loop located at each UAV that manages the bandwidth utilization. These loops control the utilization of the critical system resources and coordinate the execution of the image compression and target tracking subtasks. One approach to manage these system resources is to design *independent* feedback control loops. Unfortunately, this approach does not take into consideration the coupling between the two types of system resources and does not necessarily assure

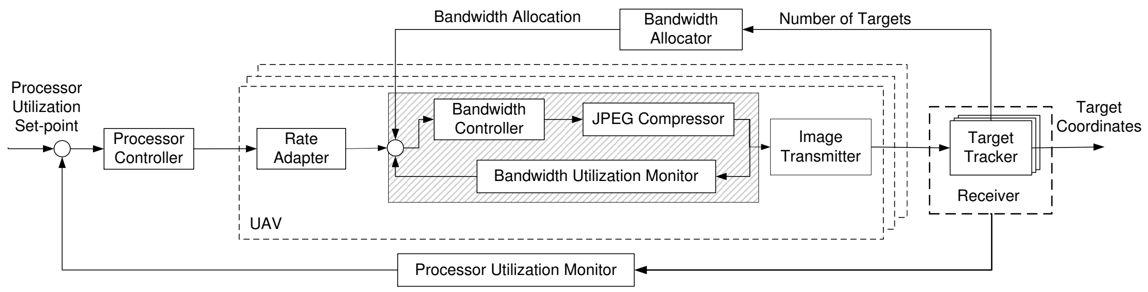


Figure 9: HiDRA's Control Architecture

system stability. Therefore, we structure these control loops in a *hierarchical* fashion so that the processor control loop at the receiver is viewed as the *outer* control loop and the bandwidth control loop at each UAV is viewed as the *inner* control loop.

As shown in Figure 10, the processor utilization monitor and processor controller serve as the resource monitor and controller of the processor control loop, respectively. The

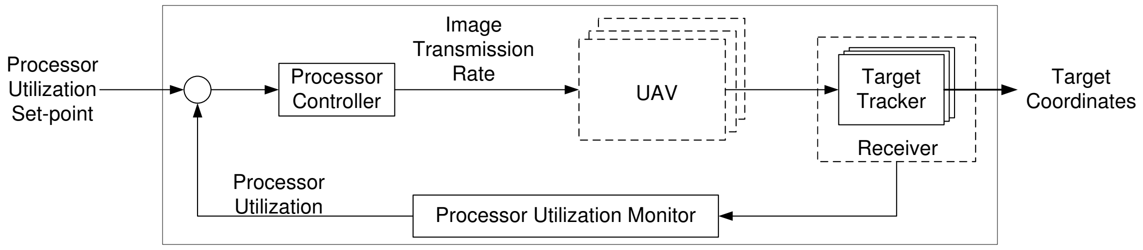


Figure 10: Processor Control Feedback Loop

objective of the processor controller is to ensure that the processor utilization is maintained at a specified set-point despite variations in resource availability and input workload. The utilization set-point of the receiver processor is an input to the processor controller and is specified during system initialization. The controlled variable for this loop is the processor utilization of the receiver, and the control input from the processor controller to the system are the image transmission rates, which are fed to the rate adapter in the UAVs. For the processor control loop, therefore, rate adapters serve as effectors.

The bandwidth allocator shown in Figure 9 is responsible for dynamically computing the bandwidth allocation to each UAV based on (1) presence/absence of objects of interest in the images received from the corresponding UAV and (2) variations in available wireless network bandwidth. The bandwidth controller of each UAV views this allocation as the bandwidth utilization set-point. The bandwidth allocator ensures that the bandwidth requirement of UAVs capturing images of one or more objects of interest is met.

As shown in Figure 11, the bandwidth utilization monitor and the bandwidth controller serve as the monitor and controller of the bandwidth control loop, respectively. The objective of the bandwidth controller is to ensure that the bandwidth utilization of the UAV is maintained at the specified set-point despite variations in resource availability and input workload. Inputs to the bandwidth controller include the bandwidth utilization set-point, which is provided by the bandwidth allocator, and image transmission rate, a model parameter of the bandwidth controller which is provided by the processor controller. Based on these inputs, the bandwidth controller computes an appropriate value of the JPEG quality factor to transmit the image of the highest quality, subjected to the specified bandwidth limitation. The controlled variable is the network bandwidth utilization of each UAV and the control input from the bandwidth controller to the system is the quality factor of the JPEG compression algorithm. This input is fed to the implementation of the JPEG compression algorithm, which serves as the effector for this control loop. The coupling between the two

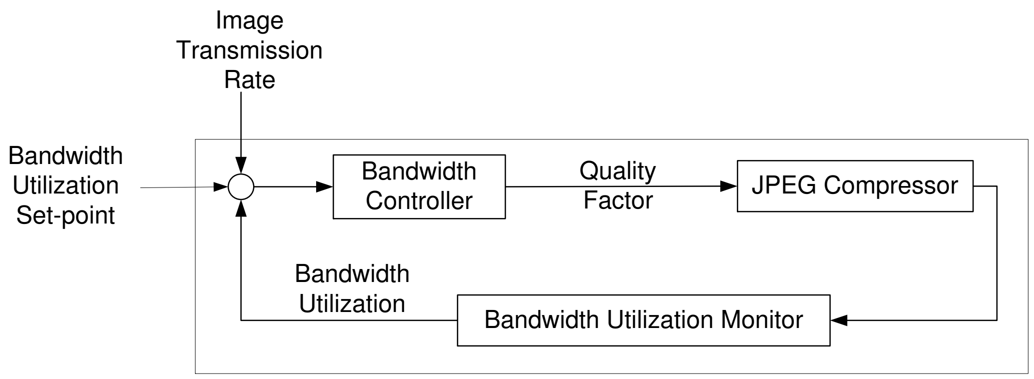


Figure 11: Bandwidth Control Feedback Loop

types of system resources is captured by using the image transmission rates computed by the processor controller as an input parameters to the bandwidth controllers.

II.5 Control Design and Analysis

This section first formalizes the resource management problem of our real-time distributed target tracking system. We then map HiDRA to this system to show how it addresses key resource management challenges of our DRE system. Finally, we present analysis that shows how HiDRA ensures the stability of our system. The formalism described below forms the foundations for the design and implementation of HiDRA. It also provides analytical assurance about system performance under fluctuating workload and varying resource availability.

II.5.1 Problem Formulation

The following notations are used throughout the remaining of the paper. The target tracking system consists of n UAVs, and therefore, n end-to-end tasks $\{T_i | 1 \leq i \leq n\}$, each with two subtasks, *i.e.*, an image compression subtask executing at UAV$_i$ and a target-tracking subtask executing at the receiver. The sampling period of the processor controller (outer feedback loop) and the bandwidth controller (inner feedback loop) are represented by T_s^{out} and T_s^{in}, respectively. The sampling periods T_s^{out} and T_s^{in} are selected to be larger than the maximum task period. All the entities that make up the bandwidth control loop (such as monitor, controller, and effector) are collocated on each UAV. However, for the processor control feedback loop, the monitor and the controller are collocated on the receiver, whereas the effectors are located at each UAV. As a result, in the processor control feedback loop, the communication between the controller and the effectors is over a wireless network. Although there are no theoretical constraints on the sampling periods, for these practical reasons, T_s^{out} is selected to be greater than T_s^{in}. In our model, k^{th} and κ^{th} sampling period represent the k^{th} sampling period of the processor controller and the κ^{th} sampling period of the bandwidth controller, respectively.

Each end-to-end task T_i is invoked periodically at a rate $r_i(k)$ at the k^{th} sampling instant of the processor controller. The rate $r_i(k)$ is assumed to take values within the range $[r_i^{min}, r_i^{max}]$. During the k^{th} sampling instant of the processor controller, images are compressed and transmitted by T_i's data source, UAV$_i$, to the receiver at the rate of $r_i(k)$ images/second. $C(k)$ represents the channel capacity (available bandwidth) of the wireless network during the k^{th} sampling period. For example, in a 802.11b wireless network, $C(k)$ can vary from 1 Mbps to 11 Mbps. The channel capacity can be obtained form the wireless network card using operating system tools/commands such as `iwlist`.

II.5.1.1 Bandwidth Allocator

During each sampling period of the processor controller, the bandwidth allocator computes a desirable bandwidth allocation for each task T_i. The wireless network bandwidth allocation to each task T_i is recomputed by the bandwidth allocator if the presence of an object of interest was detected by any of the target-tracking subtasks or a variation in the available bandwidth was detected during the previous sampling period. For each task, bandwidth is allocated such that the net bandwidth utilization is below the set-point B^s, *i.e.*:

$$\sum_{i=1}^{n} b_i^s(k) \leq B^s C(k) \tag{II.1}$$

where $b_i^s(k)$ is the bandwidth allocation (utilization set-point) for task T_i during the k^{th} sampling period of the processor controller.

Let $p(k)$ and $p_i(k)$ represent the total number of objects of interest tracked by the system and the number of objects being tracked by T_i during the k^{th} sampling period, respectively. Let b_{min} represent the minimum bandwidth allocation to each task so that images of the lowest quality can be transmitted to the receiver. Bandwidth is thus allocated to each end-to-end task as a function of $p(k)$ and $p_i(k)$ as follows:

$$b_i^s(k) = \begin{cases} B^sC(k)/n & \text{if } p(k) = 0 \\ b_{min} + \frac{(B^sC(k) - nb_{min})p_i(k)}{p(k)} & \text{if } p(k) > 0 \end{cases}, \forall \quad T_i \quad | \quad 1 \leq i \leq n. \tag{II.2}$$

If the total number of objects of interest tracked by the system is 0, bandwidth is equally allocated to each task. If the total number of objects of interest tracked by the system is greater than 0, we assume all objects of interest are of equal importance, and bandwidth allocation to tasks is based on the number of objects currently being tracked by that task. This design ensures that a greater amount of bandwidth is allocated to tasks that are currently tracking objects of interest as compared to the ones that are not. If objects of interest are of varying importance, a bandwidth allocation policy that takes into consideration the importance of object of interest can be employed without any modifications to HiDRA.

II.5.1.2 Processor Utilization Controller

We use the approach in [50] to model processor utilization. Section II.5.2 uses the following model in the stability analysis of HiDRA. The target-tracking subtask of each end-to-end task T_i has an *estimated* execution time of c_i known at design time. The estimated processor utilization by the target-tracking subtask of task T_i during the k^{th} sampling period is denoted as $E_i(k)$ and is computed as

$$E_i(k) = c_i r_i(k) \tag{II.3}$$

where $r_i(k)$ is the invocation rate of end-to-end task T_i during the k^{th} sampling period. The net estimated processor utilization during the k^{th} sampling period is therefore

$$E(k) = \sum_{i=1}^{n} c_i r_i(k). \tag{II.4}$$

At runtime, however, the *actual* execution times may be different since they depend on the presence (and number) of objects in the images. At runtime, therefore, the actual processor utilization $U(k)$ can be written as

$$U(k) = G_p(k)E(k) \tag{II.5}$$

where $G_p(k)$ is the processor utilization ratio. Although, $G_p(k)$ is unknown, it is reasonable to assume that the worst case utilization ratio $G_p = \max_k\{G_p(k)\}$ is known. Let the processor utilization set-point of the receiver node be represented as U^s. From (II.5), the process utilization model can be written as

$$\Delta U(k+1) = \Delta U(k) + G_p(k)v_p(k) \tag{II.6}$$

where $\Delta U(k) = U(k) - U^s$ and $v_p(k) = E(k+1) - E(k)$. The task of the feedback controller is to compute $v_p(k)$ so that $U(k)$ converges to U^s (or $\Delta U(k) \to 0$).

We consider a linear proportional controller

$$v_p(k) = K_p \Delta U(k) \tag{II.7}$$

where K_p is a control gain which will be selected so that the system is stable. A proportional controller is used because of the simplicity in the derivation of the control gain that ensures stability and in the implementation that incurs minimal computational overhead. Actuators implement the control signal $v_p(k)$ by changing the invocation rate of end-to-end tasks. The closed-loop system is described by

$$\Delta U(k+1) = [1 + K_p G_p(k)]\Delta U(k). \tag{II.8}$$

The control algorithm is implemented as follows. During each sampling period, the controller compares the current processor utilization $U(k)$ with the utilization set-point U^s, and computes the net estimated utilization $E(k+1)$ for the next sampling period based on the equation $E(k+1) = E(k) + K_p \Delta U(k)$. Since the presence of one or more objects of interest in the received images increases the execution time the target-tracking subtask, computational power is allocated to target tracking subtasks based on the number of objects of interest that are present in the received images. We therefore have

$$E_i(k+1) = \begin{cases} \frac{E(k+1)}{n} & \text{if } p(k) = 0 \\ E_{min} + \frac{(E(k+1) - nE_{min})p_i(k)}{p(k)} & \text{if } p(k) > 0 \end{cases}, \forall \quad T_i \quad | \quad 1 \le i \le n \qquad \text{(II.9)}$$

where $p(k)$ represents the total number of objects of interest captured by all the tasks in the system, $p_i(k)$ represents the number of objects of interest being captured by T_i during the k^{th} sampling period, and E_{min} represents the minimum processor allocation to each task so that images can be processed by the receiver at the lowest rate.

If the total number of objects of interest tracked by the system is 0, computational power is equally allocated to each task. If the total number of objects of interest tracked by the system is greater than 0, however, allocation of computational resource to tasks is weighted based on the number of objects currently being tracked by that task. This design ensures that a greater amount of computational power is allocated to tasks that are currently tracking objects of interest as compared to the ones that are not. From equations (II.3), (II.7), and (II.9) we derive the task execution rate as follows:

$$r_i(k+1) = \begin{cases} \frac{E(k)+(U(k)-U^s)K_p/G_p}{nc_i} & \text{if } p(k) = 0 \\ \frac{E_{min}}{c_i} + \frac{p_i(k)(E(k)+(U(k)-U^s)K_p/G_p - nE_{min})}{p(k)c_i} & \text{if } p(k) > 0 \end{cases}, \forall \quad T_i \quad | \quad 1 \le i \le n \qquad \text{(II.10)}$$

II.5.1.3 Bandwidth Utilization Controller

We next present the analytical model of the bandwidth controller for each UAV. The following notations are used in this model where the symbols correspond to each UAV and the subscript is omitted for simplicity:

- $b(\kappa)$: Actual bandwidth utilization in the κ^{th} sampling period.
- $b^s(k)$: Desired bandwidth utilization (set-point) computed by the bandwidth allocator in the k^{th} sampling period as shown in equation (II.2).
- $r(k)$: Task rate computed by the processor controller in the k^{th} sampling period, as shown in equation (II.10).
- s: Size of an uncompressed image, which is a constant and known at design time.
- $q(\kappa)$: Quality factor of image compression algorithm (JPEG) computed by the bandwidth controller in the κ^{th} sampling period.
- $\phi(q)$: Estimated size of the compressed image compressed with quality factor q.

To simplify our notation, we express $r(k)$ and $b^s(k)$ with respect to the index κ by defining $r(\kappa) = r(k), b^s(\kappa) = b^s(k), k \leq \kappa < k+1$.

The controlled variable of this feedback control loop is the bandwidth utilization, $b(\kappa)$, and the control input from the controller to the UAV is the quality factor of the image compression algorithm, $q(\kappa)$. The controller computes an appropriate value of quality factor, $q(\kappa)$, to ensure that the bandwidth utilization of the UAV, $b(\kappa)$, converges to the set-point, $b^s(\kappa)$, computed by equation (II.2).

The average size of the compressed image, $\phi(q)$, is related to the quality factor of the image compression algorithm, q, by a non-linear function as shown in Figure 12. For the purpose of our control design, however, we choose q within the range $[10, 70]$ where this function can be approximated by a linear one. A piecewise linear function can also be used.

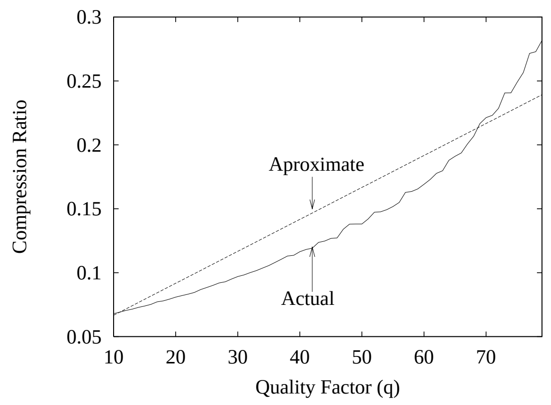


Figure 12: Linearization of $\phi(q)$

For $10 \leq q \leq 70$, we have

$$\phi(q) = sgq + \omega \tag{II.11}$$

where g is the slope and ω is the y-intersect of the linear approximation of the function in Figure 12.

Images are compressed with a quality factor q and transmitted at the rate r from the UAV to the receiver. Therefore, the bandwidth utilization contributed by the UAV is

$$\begin{aligned} b(\kappa) &= r(\kappa)\phi(q) \\ &= r(\kappa)sgq(\kappa) + r(\kappa)\omega. \end{aligned}$$

Let $\Delta b(\kappa) = b(\kappa) - b^s(\kappa)$ and $v_b(\kappa) = q(\kappa+1) - q(\kappa)$, then the bandwidth utilization can be described by the dynamical model

$$\Delta b(\kappa+1) = \Delta b(\kappa) + r(\kappa)sgv_b(\kappa). \tag{II.12}$$

The objective of the feedback controller is to determine $v_b(\kappa)$ as a function of $\Delta b(\kappa)$ so that the bandwidth utilization converges to the set-point. However, the bandwidth utilization $b(\kappa)$ is not directly available due to measurement noise. The bandwidth utilization monitor measures the bandwidth utilization as the rate at which data is written by the image compression subtask to the underlying network stack. It must be noted that the bandwidth utilization monitor measures the bandwidth utilization of the UAV and not the channel capacity or the utilization of the wireless network. Therefore, the resolution of the bandwidth utilization monitor is in the order of the size of the compressed image. Hence, even a small variation in the sampling period and the image transmission rate will considerably affect the measured bandwidth utilization. Although the sampling period of the bandwidth controller is a constant, from a practical standpoint, the sampling period might vary marginally due to the jitter associated with the timer that is employed to implement the periodic task. Moreover, the image transmission rate varies significantly at runtime since it is dynamically computed by the processor controller.

Let $\tilde{b}(\kappa)$ denote the measured bandwidth utilization in the κ^{th} sampling period. We assume that the effect of the measurement noise can be described by

$$\tilde{b}(\kappa) = b(\kappa) + n(\kappa)$$

where the measurement noise $n(\kappa)$ is assumed to be a discrete-time Gaussian process with zero mean and variance $E[n^2(\kappa)]$. The variance can be approximated experimentally by transmitting images with a known rate and computing the square of the rms value of the difference between the predicted utilization $b(\kappa)$ and the measured utilization $\tilde{b}(\kappa)$ [31].

To remove the measurement noise in the measured bandwidth utilization, we employ a Kalman filter [85] to estimate the actual bandwidth utilization. Alternatively, a simple low-pass measurement filter can be used. We select a Kalman filter because it provides good transient and steady-state performance, it is optimal in the sense that the variance of the estimation error is minimized, and it allows the stability analysis of the closed loop system based on the certainty equivalence principle (or separation principle) [6]. It should be noted that the processor utilization monitor obtains the processor utilization directly from the underlying operating system, and therefore is of higher resolution compared to the bandwidth utilization monitor.

The Kalman filter computes recursively the estimated bandwidth utilization $\hat{b}(\kappa)$ based on the measured bandwidth utilization $\tilde{b}(\kappa)$ and the bandwidth utilization model (II.12). Let $\hat{b}^{-}(\kappa)$ the predicted bandwidth utilization in the κ^{th} sampling period given by

$$\hat{b}^{-}(\kappa) = \hat{b}(\kappa-1) + r(\kappa-1)sgv_b(\kappa-1).$$

The output of the Kalman filter is

$$\hat{b}(\kappa) = \hat{b}^{-}(\kappa) + K(\kappa)(\tilde{b}(\kappa) - \hat{b}^{-}(\kappa)) \tag{II.13}$$

where $K(\kappa)$ is a filter gain that is computed recursively in order to minimize the variance of the estimation error $\varepsilon(\kappa) = b(\kappa) - \hat{b}(\kappa)$ [6, 85].

The output of the Kalman filter, $\hat{b}(\kappa)$, is used by the bandwidth controller as the current bandwidth utilization. We consider a linear controller

$$v_b(\kappa) = K_b \Delta\hat{b}(\kappa) \tag{II.14}$$

where K_b is the control gain that will be selected so that the system is stable. During each sampling period, the controller compares the estimated bandwidth utilization $\hat{b}(\kappa)$ with the utilization set-point $b^s(\kappa)$, and computes the quality factor $q(\kappa+1)$ by

$$q(\kappa+1) = q(\kappa) + K_b \Delta\hat{b}(\kappa). \tag{II.15}$$

II.5.2 Stability Analysis

A control system is said to be stable if and only if the system converges to an equilibrium for any set of initial conditions. In our case study, the initial conditions are used to represent the changes in workload (due to the change of the images' content) and/or resource availability. Our target tracking system is therefore stable if resource utilization of both the system resources (*i.e.*, processor utilization at the receiver and the network bandwidth utilization) converge to their respective utilization set-points in the presence of workload changes and/or resource availability. Although the controller is designed based on a time-invariant model (constant upper bounds on resource utilization), we show that the system is stable even when resource availability and/or utilization changes at runtime, *i.e.*, the system is time-varying.

A feedback control loop can be stabilized by selecting the controller so that the poles of the closed loop system are in the unit circle [6, 31]. The bandwidth utilization control loop includes the Kalman filter (II.13) and the linear controller (II.14). A consequence of the separation principle is that the control synthesis problem can be solved separately and the dynamics of the closed-loop system are determined by the dynamics of the controller and the optimal filter [6]. Specifically, the poles of the closed loop system are determined by the poles of the controller and the poles of the Kalman filter. At steady-state the gain of the Kalman filter converges to a stationary value that ensures stability for the estimation error $\varepsilon(\kappa)$. Therefore, in our analysis we can focus on imposing conditions on the bandwidth utilization control gain K_b to ensure that the pole of the bandwidth utilization controller is inside the unit circle.

We can stabilize each of the two types of feedback control loops by selecting the gains K_p and K_b so that the corresponding poles are in the unit circle. Such a design, however, does not necessarily ensure the stability of the hierarchical control architecture since it does not take into consideration the interaction between the feedback loops (due to the presence of $r(\kappa)$ in equation (II.12)). We next present an analysis result that allows us to select the control gains so that the overall stability is assured.

Assuming that the input buffer of the receiver is never empty, it is clear that the processor utilization is independent of the bandwidth utilization. If we select K_p so that

$-2/G_p < K_p < 0$ then

$$\Delta U(k) = [1 + K_p G_p(k)]^k \Delta U(k_0), k \geq k_0$$

and $\Delta U(k) \rightarrow 0$ since $|1 + K_p G_p(k)| < 1$.

From equation (II.10), it follows that in the steady state the utilization for each task $U_i(k)$ will be stable (it will converge to a set-point U_i^s that depends on the presence of objects in the image data) and we can write

$$\Delta U_i(k+1) = \alpha_i(k)\Delta U_i(k) \tag{II.16}$$

where the function $\alpha_i(k)$ satisfies $|\alpha_i(k)| < 1$.

Let r_i^s denote the rate of the i^{th} task at the steady state, then $r_i(k) = r_i^s + \Delta r_i(k)$ where $\Delta r_i(k) \rightarrow 0$. The bandwidth utilization model for the i^{th} UAV is

$$\Delta b_i(\kappa+1) = [1 + (r_i^s + \Delta r_i(\kappa)) sgK_b^i]\Delta b_i(\kappa) \tag{II.17}$$

The primary challenge of the stability analysis of our framework is the coupling between the processor and bandwidth controllers. As it can be seen in equation (II.17), the control input from the processor controller to the system, $\Delta r_i(\kappa)$, is used by the bandwidth controller. Our objective is to deduce the stability properties of the system (II.16-II.17) by studying the *isolated system*

$$\Delta U_i(k+1) = \alpha_i(k)\Delta U_i(k) \tag{II.18}$$

$$\Delta b_i(\kappa+1) = [1 + r_i^s sgK_b^i]\Delta b_i(\kappa) \tag{II.19}$$

where the equations have been decoupled by setting $\Delta r_i(\kappa) = 0$.

Theorem 1. *The system (II.16-II.17) is stable if and only if the isolated system (II.18-II.19) is stable.*

Proof. Define the norm $||[x_1,x_2]|| = ||[x_1,x_2]||_\infty = \max\{|x_1|,|x_2|\}$ and denote $\Delta U_i(k), \Delta b_i(\kappa)$ and $\Delta U_i^I(k), \Delta b_i^I(\kappa)$ the solutions of (II.16-II.17) and (II.18-II.19) respectively.

"Only-if": If the system (II.16-II.17) is stable, then there exists function $\alpha(\kappa)$ with $\alpha(\kappa) \to 0$ such that

$$||[\Delta U_i(\kappa), \Delta b_i(\kappa)]^T|| \le \alpha(\kappa)||[\Delta U_i(\kappa_0), \Delta b_i(\kappa_0)]^T|| \tag{II.20}$$

$\forall \kappa \ge \kappa_0$ and for every initial condition $[\Delta U_i(\kappa_0), \Delta b_i(\kappa_0)]^T$ where $\Delta U_i(\kappa) = \Delta U_i(k), k \le \kappa < k+1$.

In particular, suppose that the initial condition is $[0, \Delta b_i(\kappa_0)]^T$, then by equation (II.20) $\forall \kappa \ge \kappa_0$, $|\Delta b_i^I(\kappa)| \le \alpha(\kappa)|\Delta b_i^I(\kappa_0)|$, which shows that the system (II.18-II.19) is stable.

"If": It is easy to see that $\Delta U_i(k) = \Delta U_i^I(k)$ so we have to analyze only $\Delta b_i(\kappa)$. Define $\eta_I(\kappa) = 1 + r_i^s g K_b^i$ and $\eta(\kappa, \Delta r_i(\kappa)) = 1 + (r_i^s + \Delta r_i(\kappa)) g K_b^i$. From the stability of (II.18-II.19), we have that $|\eta_I(\kappa)| < 1$ and there exists a function $\alpha_2(\kappa)$ with $0 \le \alpha_2(\kappa) \to 0$ such that

$$\Delta b_i^2(\kappa)(\eta_I^2(\kappa) - 1) \le -\alpha_2(\kappa)\Delta b_i^2(\kappa_0)$$

for every $\Delta b_i(\kappa_0)$ and $\kappa \ge \kappa_0$. But we can write

$$\begin{aligned}\Delta b_i^2(\kappa+1) - \Delta b_i^2(\kappa) &= \Delta b_i^2(\kappa)(\eta_I^2(\kappa) - 1) + \Delta b_i^2(\kappa)(\eta^2(\kappa, \Delta r_i(\kappa)) - \eta_I^2(\kappa)) \\ &\le -\alpha_2(\kappa)\Delta b_i^2(\kappa_0) + \gamma(\kappa)\end{aligned}$$

where $\gamma(\kappa) \to 0$ since $\Delta r_i(\kappa) \to 0$. $\Delta b_i(\kappa) \to 0$ and the system (II.16-II.17) is therefore stable. □

Using the above theorem, we can select the control gains so that our hierarchical control architecture is stable. For the processor utilization feedback loop, the gain could be selected to satisfy $-2/G_p < K_p < 0$ that ensures stability [3, 50]. Similarly, for the bandwidth utilization control loop, the gain should be selected so that (II.19) is stable. Since r_i^s is not known at design time, we can select the gain to satisfy $-2/(r_i^{\max}) < K_b^i < 0$. A reasonable choice for selecting the control gains is to use deadbeat control [31] based on the worst case utilization ratio and maximum task rate respectively, i.e. $K_p = -1/G_p$ and $K_b^i = -1/r_i^{\max}$. This selection tries to minimize the settling time keeping the overshoot equal to zero. Although deadbeat control may introduce saturation if the ranges for the control effectors, i.e. the rate and the quality factor are small, its performance was satisfactory for our case study. Other criteria for selection of the gain can be found in [50].

II.6 Performance Results and Analysis

This section first presents the testbed for our target tracking system, which was used to evaluate the performance of HiDRA in the context of a representative open DRE system. We then describe our experiments and analyze the results obtained to evaluate the performance of our DRE system empirically with and without HiDRA under varying wireless bandwidth availability and input workload. The goal of our experiments was to validate our theoretical claims and show that HiDRA yields predictable and high-performance resource management and coordination for multiple types of resources.

II.6.1 Hardware and Software Testbed

Our experiments were performed on the Emulab [87] testbed at University of Utah (`www.emulab.net`). The hardware configuration consists of three nodes acting as UAVs and one receiver node. Images from the UAVs were transmitted to a receiver via a wireless LAN configured with a maximum channel capacity of 2 Mbps. The hardware configuration of all the nodes was a 3 GHz Intel Pentium IV processor, 1 GB physical memory, 802.11 a/b/g WIFI interface (Atheros 5212 chipset), and 120 GB hard drive. The Redhat 9.0 operating system with wireless support was used for all the nodes.

The following software packages were also used for our experiments: (1) **TAO 1.4.7**, which is our open-source implementation of Real-time CORBA [62] that HiDRA and our DRE system case study are built upon, (2) **Ffmpeg 0.4.9-pre1** with **Fobs-0.4.0** front-end, which is an open-source library that decodes video encoded in MPEG-2, MPEG-4, Real Video, and many other video formats to yield raw images, and (3) **ImageMagick 6.2.5**, which is an open-source software suite that we used to compress the raw images to JPEG image format.

II.6.2 Target Tracking DRE System Implementation

The entities in our target tracking DRE system are implemented as CORBA objects and communicate over the *TAO* [67] Real-time CORBA Object Request Broker to achieve desired real-time performance. The end-to-end application consists of pairs of CORBA objects: the UAV data source and the receiver data sink. The UAV data source object that executes on each UAV's on-board processor performs the following actions: (1) extracts raw images from an on-disk video file using Ffmpeg with Fobs front end[1], (2) compress the raw image into JPEG format using ImageMagick, and (3) "pushes" the compressed images over the wireless link to the data sink object via a CORBA oneway method invocation.

A data sink object at the receiver processes the images received from the corresponding UAV. Each data sink object contains two functional modules: one that determines the presence of one or more objects of interest in the received images, and the other tracks the coordinates of objects of interest in the received image, if present. The second functional module is executed only if the presence of one or more objects of interest is detected by the first module.

To perform target tracking, received images are compared with a reference image, that is given during system initialization. To obtain the reference image, a raw image is extracted from a frame in the video that contains the object of interest. This raw image is then compressed using JPEG compression algorithm with a quality factor of 100 and used as

[1]We used pre-recorded video which was made available on each UAV node as our source of "live" video.

the reference image. The received images are converted from color to gray-scale, and the processed image is “subtracted” from the reference image to obtain the difference image. If the average pixel value of the difference image is greater than a threshold (which indicates the presence of one of more objects of interest), the center of mass of the objected is computed. This approach is common and the coordinates of a moving object can be tracked using a Kalman filter [26].

Table 2 summarizes the number of lines of code of various entities in our middleware and DRE multimedia system case study.[2]

Entity	Total Lines of Source Code
HiDRA	12,243
DRE Target Tracking System	19,875
Ffmpeg + Fobs	214,092
ImageMagick	253,270
The ACE ORB (TAO)	907,035

Table 2: Lines of Source Code for Various System Elements

II.6.3 Experiment Configuration

Our experiments consisted of three (emulated) UAVs containing the data source object that (1) decoded the video from a file, (2) extracted the raw images, (3) compressed them using JPEG compression, and (4) transmitted the compressed images to the corresponding data sink object at the receiver node. Wireless network bandwidth was shared between the three data source/data sink object pairs, and the computational power at the receiver node was shared between the three data sink CORBA objects.

We evaluated the adaptive resource management capabilities of HiDRA under the following operational conditions: (1) constant bandwidth availability and constant workload,

[2]Lines of source code was measured using SLOCCount (`http://www.dwheeler.com/sloccount/`).

(2) constant bandwidth availability and varying workload, (3) varying bandwidth availability and constant workload, and (4) varying bandwidth availability and varying workload. These experimental configurations were chosen to evaluate the performance of HiDRA under all possible combinations of fluctuations in bandwidth availability and input workload. We evaluate the performance of the system when it was operated with independent feedback control loops to demonstrate the advantages of the proposed hierarchical architecture. In all operating conditions, we monitored the processor utilization at the receiver and wireless network bandwidth utilization between the UAVs and the receiver. Processor utilization at each UAV node was not monitored since the computational power of the UAV on-board processor was sufficiently large to compress images of the highest quality and resolution and transmit them to the receiver without overloading the processor.

Bandwidth consumption by each UAV was measured as the rate at which data was written to the underlying network stack by the UAV data source CORBA object. The bandwidth utilization can also be measured at the receiver node using the techniques described in [69]. Since our measurement of bandwidth consumption by each UAV was noisy, we used a Kalman filter to suppress the disturbances in the measured bandwidth utilization. Processor utilization at the receiver was measured using the data from the `/proc/stat` file. In our experiments, we also measured application QoS properties, such as target-tracking precision and average end-to-end delay.

We defined target-tracking precision as the inverse of *target-tracking error*, which is the distance between the computed center of mass of an object and the actual center of mass of the object. To compute the actual center of mass of the object, we identified an object present in the video as the object of interest, performed target-tracking on the raw images extracted from the video, and used this value as a reference. At the data sink object, the target-tracking results were then compared with this reference value.

End-to-end delay consists of (1) processing delay at the UAV, (2) network transmission delay from the UAV node to the receiver and (3) processing delay at the receiver node. To measure the end-to-end delay, an image was timestamped by the data source object when the raw image was extracted from the pre-recorded video file, before it was compressed and transmitted to the corresponding data sink object. Upon completion of processing of the received image by the data sink object, the time-stamp of the image was compared with the current time on the receiver node to obtain the end-to-end delay. To eliminate time skews, physical clocks on all the nodes in our hardware testbed were synchronized using NTP [55].

In all the above listed operational conditions, we compare the performance of our DRE system when it was operated with and without HiDRA. Comparison of system performance is decomposed into comparison of resource utilization and application QoS. For system resource utilization, we compare (1) wireless network bandwidth utilization and (2) processor utilization of the receiver node. For application QoS, we compare (1) target-tracking precision and (2) average end-to-end delay.

For all our experiments, we chose the sampling period of the processor controller and the bandwidth controller as 10 seconds and 1 second, respectively. The minimum and maximum image transmission rate $[r_{min}, r_{max}]$ was 5 and 15 images/second. Therefore, as explained in Section II.5.2, the control gain for the bandwidth controller (K_b) was computed to be -0.06 ($-1/15$). Since G_p was measured to be 2, the control gain for the processor controller (K_p) was computed to be -0.5 ($-1/2$). The processor utilization set-point was selected to be 0.7. The goal of utilization control is to (1) prevent processor overload (which can cause system instability), and (2) avoid unnecessarily under utilizing the processor (which leads to a low task rate). The choice of 0.7 as the set point achieves the desired trade off between overload protection and high task rate in our system. Since an IEEE 802.11 DCF-based network has a utilization of approximately 0.7 with 20 active nodes [9], the wireless bandwidth utilization set-point was also configured at 0.7. Although for a system with four nodes the achievable channel utilization could be higher than 0.7 (e.g., as high as 0.8), this value varies depending on many other factors such as packet size, channel bit rate, etc. Considering all these factors, we set the bound to 0.7 conservatively.

II.6.4 Experiment 1 : Constant Bandwidth Availability and Constant Workload

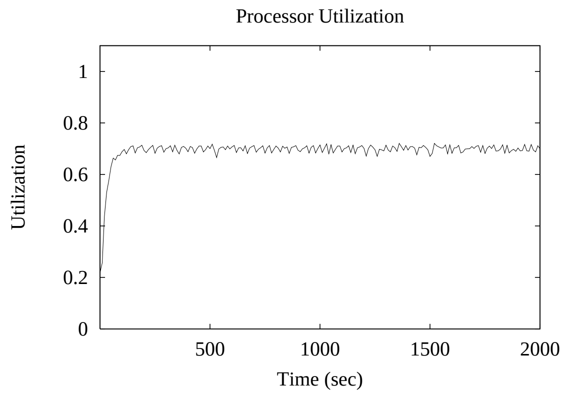
We now present the results obtained from running the experiment under a constant channel capacity of 2 Mbps and a constant 2 objects of interest tracked by the system. This experimental setup provides an operational condition where resource availability and input workload are known *a priori* and not subjected to change during the course of the experiments. Images containing objects of interest were captured by UAVs 1 and 2. This experiment serves as the baseline for all other experiments. It validates that when the tracking system is operated with HiDRA the following behavior occurs: (1) utilization of system resources converge to their respective set-points and (2) application QoS converges to the values that were obtained when the system was operated without HiDRA and application parameters were chosen *a priori*.

UAV	Image Transmission Rate (images/sec)	Quality Factor
UAV 1	10	40
UAV 2	10	40
UAV 3	10	40

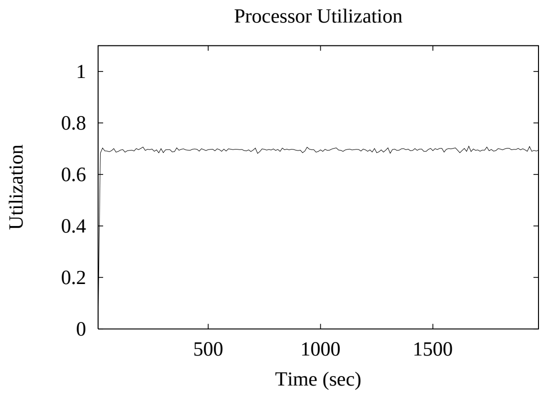
Table 3: Application Parameters Chosen in Advance

We compare the performance against a static configuration. In the static configuration, application parameters, such as image transmission rates and quality factor of the JPEG image compression algorithm, were chosen *a priori*. Values of these parameters were selected such that (1) both processor utilization of the receiver node and the wireless bandwidth utilization is equal to the set-point of 0.7 and (2) application QoS are maximized. The settings of the static configuration of the system are shown in Table 3.

II.6.4.1 Comparison of Resource Utilization



(a) Processor Utilization with HiDRA



(b) Processor Utilization without HiDRA

Figure 13: Exp 1: Comparison of Processor Utilization

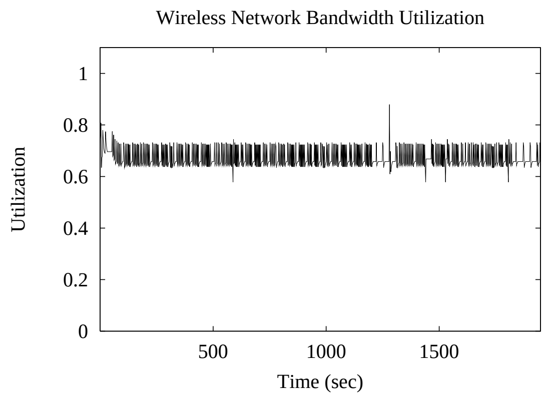
Figures 13 and 14 compare the processor utilization at the receiver node and the wireless network bandwidth utilization when the system was operated with and without HiDRA. The output of the bandwidth utilization monitor, shown in Figure 14b, was processed with a Kalman filter and used by the bandwidth controller as the current bandwidth utilization.

Figures 13b and 14c show that when the system was operated without HiDRA, resource utilization of both the resources is 0.7 during the course of the experiment. Similarly, Figures 13a, 14a, and 14b show that when the system was operated with HiDRA, resource utilization converges to the set-point of 0.7 and in maintained at 0.7 for the remaining duration of the experiment. These results show that when the system is operated using HiDRA, system resource utilization converges to the respective utilization set-points.

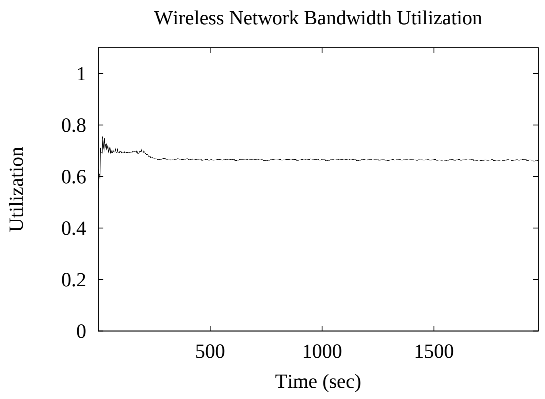
II.6.4.2 Comparison of QoS

We now compare the application QoS – (1) target-tracking precision, and (2) average end-to-end delay.

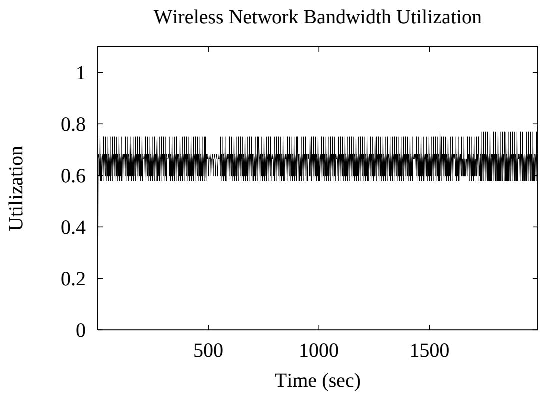
Figure 15 compares the target-tracking error obtained when the system was operated



(a) Bandwidth Utilization with HiDRA



(b) Bandwidth Utilization with HiDRA (estimates using a Kalman Filter)



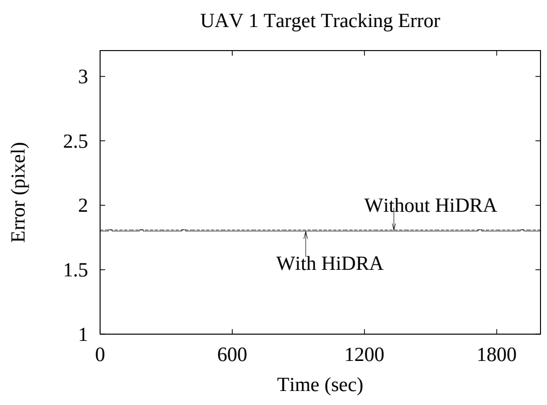
(c) Bandwidth Utilization without HiDRA

Figure 14: Exp 1: Comparison of Bandwidth Utilization

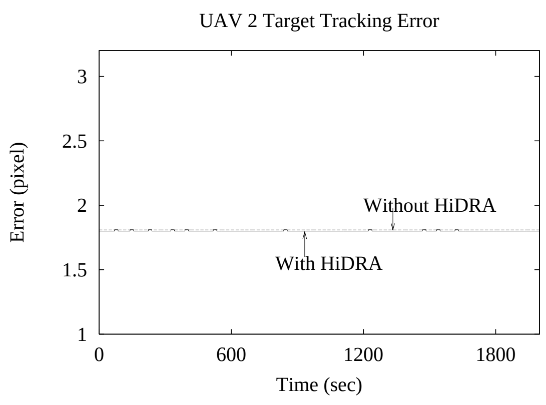
with and without HiDRA. Figures 15a and 15b show that average target-tracking error—and therefore target-tracking precision—is nearly the same when the system was operated with and without HiDRA.

Table 4, which compares the end-to-end delay when the system was operated with and without HiDRA, shows that average end-to-end delay is the same as when the system was operated with and without HiDRA. Based on these results, we conclude that QoS of applications in our DRE system converges to the values obtained when the system was operated without HiDRA and application parameters were chosen *a priori*.

From our comparison of resource utilization and system QoS, we conclude that when



(a) UAV-1



(b) UAV-2

Figure 15: Exp 1: Comparison of Target-tracking Error

Number of Objects	End-to-End Delay (msec)	
	With HiDRA	Without HiDRA
2	117	117

Table 4: Exp 1: Comparison of End-to-End Delay

the system is operated with HiDRA (1) utilization of system resources converge to their respective set-points and (2) application QoS converge to the values that were obtained when the system was operated without HiDRA and application parameters were chosen *a priori*.

II.6.5 Experiment 2: Decoupled Independent Feedback Control Loops

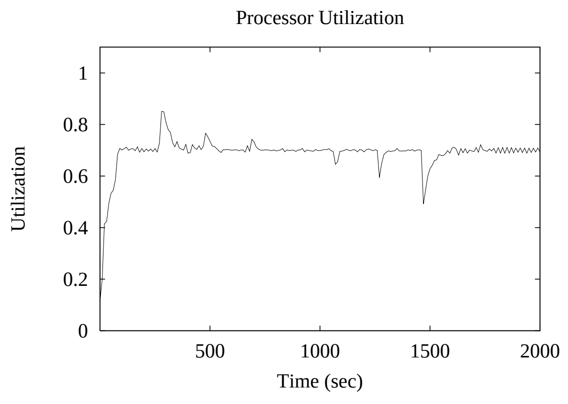
We now demonstrate the effect of employing the processor control loop and bandwidth control loops in an independent fashion. To decouple these two types of feedback control loops, the bandwidth controller of all the UAVs assume a constant image transmission rate of 10 images per second. However, the actual image transmission rates are dynamically modified by the processor controller and the rate adapter at runtime.

In this section, we present the results obtained from running the experiment under a constant channel capacity of 2 Mbps and varying number of objects of interest in the system. This experiment demonstrates the need for an hierarchical architecture by analyzing the effect of employing multiple independent feedback loops under constant resource availability and varying input workload. Table 5 summarizes the number of objects of interest that were tracked as a function of time.

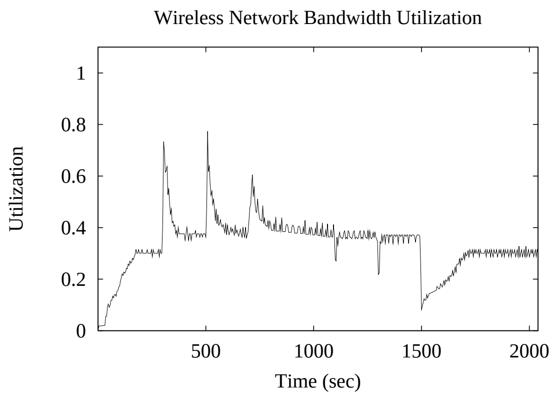
Time (sec)	Number of Objects			
	UAV 1	UAV 2	UAV 3	Total
0 - 300	0	0	0	0
300 - 500	1	0	0	1
500 - 700	1	1	0	2
700 - 1,100	1	1	1	3
1,100 - 1,300	0	1	1	2
1,300 - 1,500	0	0	1	1
1,500 - 2,000	0	0	0	0

Table 5: Objects of Interest as a Function of Time

II.6.5.1 Analysis of Resource Utilization



(a) Processor Utilization



(b) Bandwidth Utilization

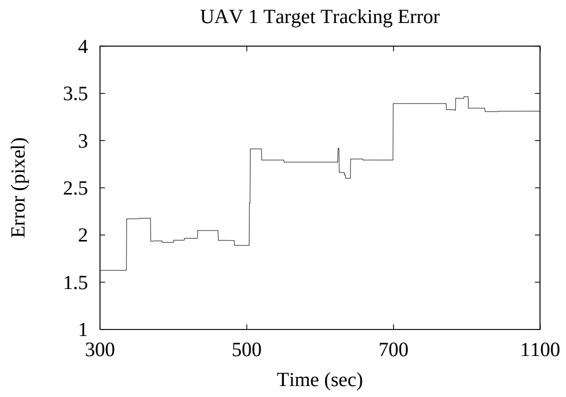
Figure 16: Exp 2: Resource Utilization

Figure 16a shows the processor utilization at the receiver node when the system was operated with independent feedback loops. Figure 16a and Table 5 show that the increase in the processor utilization at $T = 300s$ is due to the presence of the first object of interest. Figure 16a shows that although the processor utilization increased above 0.7, within the next several sampling periods, the processor control loop restored the processor utilization to the desired set-point of 0.7. This was achieved as a result of reducing the execution rates of data-source/receiver pair(s) deemed less important, *i.e.*, ones that captured images where objects of interest were absent. At $T = 500s$ and $T = 700s$, the presence of the second and third object of interest were detected. As Figure 16a shows, the processor utilization quickly re-converges to the set-point after a transient increase. At $T = 1,100s$ the total number of objects being tracked by the system reduced from 3 to 2. Although there was a decrease in the processor utilization, the processor control loop restored the processor utilization to the set-point by increasing the execution rate of important data-source/data sink pair(s). Similarly, the processor control loop ensured that the processor utilization converges to the desired set-point for the remaining duration of the experiment.

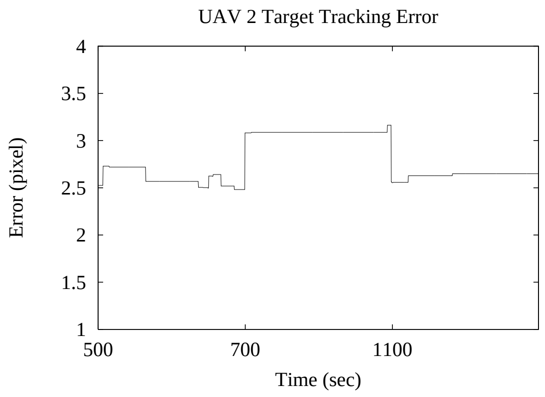
From Figure 16b, which shows the wireless network bandwidth utilization when the system was operated with independent feedback loops, it can be seen that the bandwidth utilization is significantly below the set-point of 0.7 during the entire course of the experiment. This is because the bandwidth controller assumes that the image transmission rate to be a constant 10 images per second, where as the image transmission rate is dynamically varied by the processor controller and the rate adapter at runtime in order to maintain the processor utilization at the desired value of 0.7. The bandwidth controller does not have complete knowledge of the state of the system, namely the image transmission rate, and as a result, the quality factor computed by the bandwidth controller does not aid the UAV in achieving the desired bandwidth utilization.

II.6.5.2 Analysis of QoS

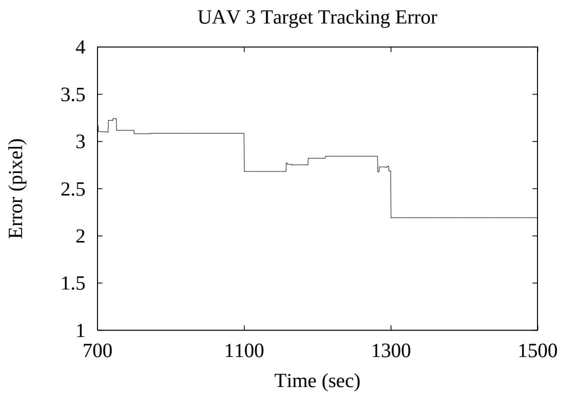
We now analyze the application QoS – (1) target-tracking precision and (2) average end-to-end delay. From Figure 17, which shows the target-tracking errors that was obtained



(a) UAV-1



(b) UAV-2



(c) UAV-3

Figure 17: Exp 2: Target-tracking Error

when the system was operated with independent feedback loops, it can be seen that the target tracking error is high when the system was operated with independent feedback loops. This is because the bandwidth control loops compute images quality factors that do not utilize the bandwidth allocated to each UAV effectively. Therefore, as shown in Figure 16b, the wireless network bandwidth was severely under-utilized. This accounts for the high target tracking error when the system was operated with independent feedback loops.

These results demonstrate that when the system was operated with independent feedback loops, wireless network bandwidth was severely under-utilized, which therefore leads to a high target tracking error, or a low QoS.

Number of Objects	End-to-End Delay (msec)
0	20
1	60
2	117
3	157

Table 6: Exp 2: End-to-End Delay

Table 6 shows the end-to-end delay when the system was operated with independent feedback loops. From Tables 4 and 6 it can be seen that when the system tracked 2 objects of interest, the same end-to-end delay was achieved when the system was operated with independent feedback loops, with HiDRA, and without HiDRA as the system resource utilization was maintained below the specified utilization set-point. This is because the wireless network begins to experience packet losses and re-transmissions when the utilization is above 0.7 [9]. When the system was operated with HiDRA, without HiDRA, and with independent feedback loops, since the bandwidth utilization was below 0.7, the network transmission delays are nearly equal. Moreover, since the processor utilization in both the cases were below the utilization set-point (as shown in Figures 13a, 13b and 16a), the end-to-end delays are equal.

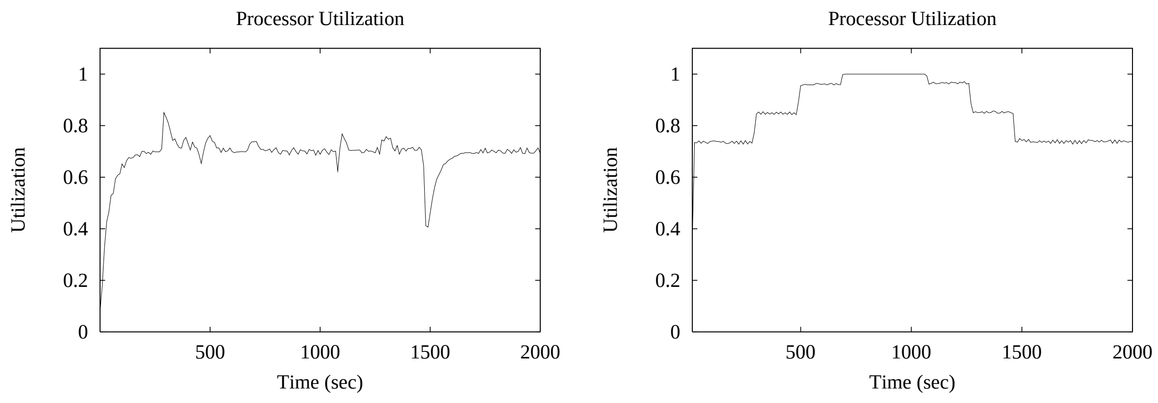
These results show the effect of employing multiple feedback loops— processor control loop and bandwidth control loops—in an independent fashion. Although the processor utilization converges to the desired value, the bandwidth utilization is significantly lower than the desired value. This results in severe under utilization of system resources and low

QoS, both of which are undesirable. Therefore, we now demonstrate how HiDRA, using an hierarchical approach, achieves desired system resource utilization and improves QoS.

II.6.6 Experiment 3: Constant Bandwidth Availability and Varying Workload

We next present the results obtained from running the experiment under a constant channel capacity of 2 Mbps and varying number of objects of interest in the system. This experiment demonstrates the adaptive resource management capabilities of HiDRA under constant resource availability and varying input workload. Table 5 summarizes the number of objects of interest that were tracked as a function of time. In this experiment, when the system was operated without HiDRA, the static system configuration shown in Table 3 was used.

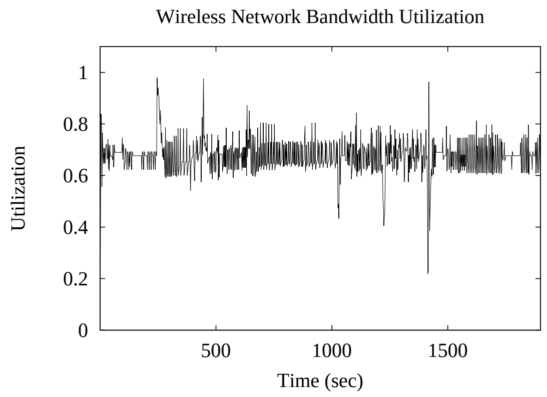
II.6.6.1 Comparison of Resource Utilization



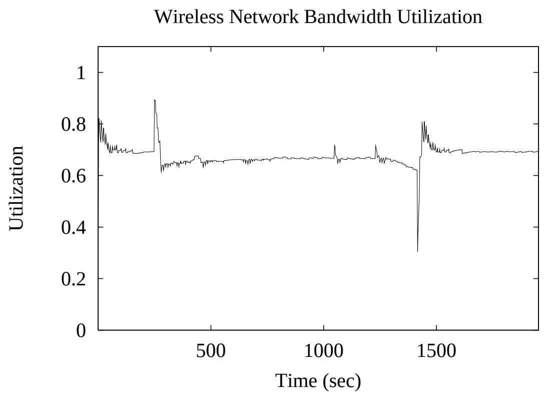
(a) Processor Utilization with HiDRA (b) Processor Utilization without HiDRA

Figure 18: Exp 3: Comparison of Processor Utilization

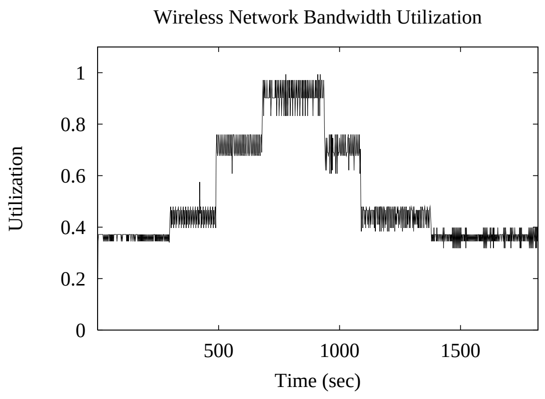
Figures 18 and 19 compare the processor utilization at the receiver node and the wireless network bandwidth utilization when the system was operated with and without HiDRA.



(a) Bandwidth Utilization with HiDRA



(b) Bandwidth Utilization with HiDRA (processed using a Kalman Filter)



(c) Bandwidth Utilization without HiDRA

Figure 19: Exp 3: Comparison of Bandwidth Utilization

The output of the bandwidth utilization monitor, shown in Figure 19b, was processed with a Kalman filter and used by the bandwidth controller as the current bandwidth utilization.

Figures 18a and 18b and Table 5 show that the increase in the processor utilization at $T = 300s$ is due to the presence of the first object of interest. Figure 18a shows that although the processor utilization increased above 0.7, within the next several sampling periods, HiDRA restored the processor utilization to the desired set-point of 0.7. HiDRA achieved this result by reducing the execution rates of data-source/receiver pair(s) deemed less important, *i.e.*, ones that captured images where objects of interest were absent. As shown in Figure 18b, when the system was operated without HiDRA, the processor utilization remained at 0.85, which is significantly higher than the utilization set-point of 0.7.

At $T = 500s$, the presence of the second object of interest was detected. The processor utilization thus increased to 0.9 when the system was operated without HiDRA, as shown in Figure 18b. As Figure 18a shows that the processor utilization quickly re-converges to the set-point after a transient increase when the system was operated with HiDRA.

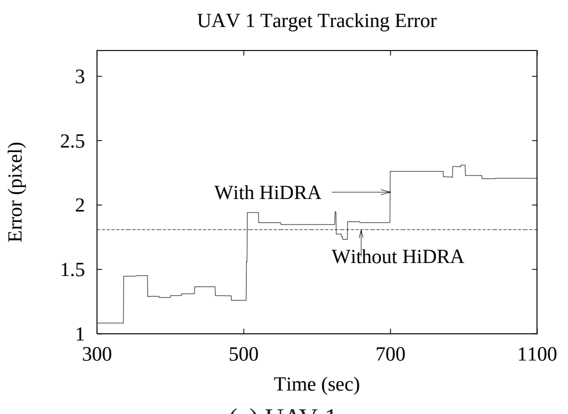
At $T = 700s$, the presence of the third object of interest was detected. As a result, when the system was operated without HiDRA, the processor utilization increased to 1, as shown in Figure 18b. Once again, Figure 18a shows that the processor utilization quickly re-converges to the set-point after a transient increase when the system was operated with HiDRA.

At $T = 1,100s$ the total number of objects being tracked by the system reduced from 3 to 2. Although there was a decrease in the processor utilization, HiDRA restored the processor utilization to the set-point by increasing the execution rate of important data-source/data sink pair(s). Similarly, HiDRA ensured that the processor utilization converges to the desired set-point for the remaining duration of the experiment. Similarly, Figures 19a and 19b shows how HiDRA ensures that the wireless bandwidth utilization converges to the desired set-point of 0.7 within bounded time, even under fluctuating workloads.

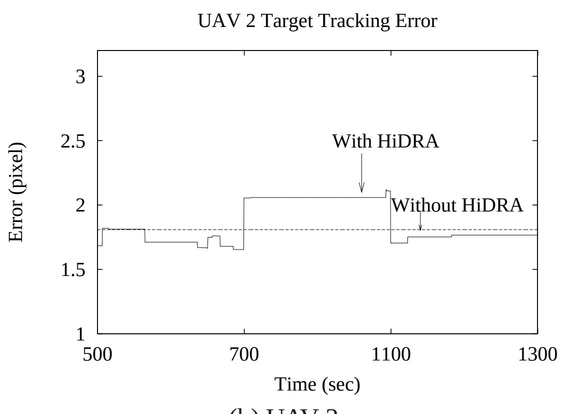
These results show how HiDRA ensures that the processor utilization of the receiver node—as well as the wireless bandwidth of the network—converges to the desired set-point within bounded time, even under fluctuating workloads. We therefore conclude that HiDRA ensures utilization of multiple system resources is maintained within the specified bounds, thereby ensuring system stability.

II.6.6.2 Comparison of QoS

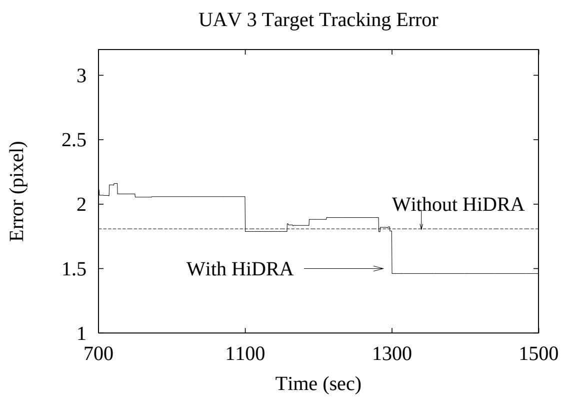
We now compare the application QoS – (1) target-tracking precision and (2) average end-to-end delay.



(a) UAV-1



(b) UAV-2



(c) UAV-3

Figure 20: Exp 3: Comparison of Target-tracking Error

Figure 20 compares the target-tracking errors that were obtained when the system was operated with and without HiDRA. Table 5 shows that during $T \in [300s, 500s]$, there was only one object of interest that was tracked by the system, and this object was tracked by UAV 1. When the system was operated without HiDRA, the static configuration of the system (shown in Table 3) assumed that there was a total 2 objects of interests being tracked by the system. As a result, the Figure 20a shows that the target tracking error during $T \in [300s, 500s]$ is lower when the system was operated with HiDRA than without it.

During $T \in [500s, 700s]$, a total of 2 objects of interest that were tracked by the system, and these objects were tracked by UAV 1 and UAV 2. This input workload is the same as the static configuration of the system. As a result, Figures 20a and 20b show that the target tracking error during $T \in [500s, 700s]$ is nearly the same when the system was operated with and without HiDRA.

During $T \in [700s, 1100s]$, however, a total of three objects of interest were being tracked by the system, one by each UAV. This input workload is higher than the input workload under which the static configuration of the system was selected. To maintain the bandwidth utilization within specified bounds, therefore, HiDRA lowers the quality factor of the images transmitted by the UAVs to the receiver during $T \in [700s, 1100s]$. As a result, Figures 20a, 20b, and 20c show that the target tracking error during $T \in [700s, 1100s]$ is higher when the system was operated with HiDRA than without it. Similarly, the target tracking precision of the received images for the remaining time intervals can be analyzed.

These results demonstrate that HiDRA effectively maintains utilization of system resource below the specified set-points despite fluctuations in input workload by gracefully adjusting application QoS.

Number of Objects	End-to-End Delay (msec)	
	With HiDRA	Without HiDRA
0	20	20
1	60	60
2	117	117
3	160	250

Table 7: Exp 3: Comparison of End-to-End Delay

Table 7 compares the end-to-end delay when the system was operated with and without HiDRA. This table shows that when the total number of objects of interest tracked by the system was 2 or less, the end-to-end delay was the same when the system was operated with and without HiDRA. This result occurred because the static configuration of the system was selected assuming 2 objects of interest were being tracked by the system. When the number of objects tracked by the system increased to 3, however, system resource were over-utilized considerably when the system was operated without HiDRA, as compared to when the system was operated with it. As a result, when the system was operated without HiDRA, the end-to-end delay is significantly higher than when the system was operated with HiDRA.

HiDRA reacts to fluctuations in input workload by modifying application parameters such as JPEG quality factor. These adaptations ensure that system resources are not over-utilized and thus lowers average end-to-end delay.

II.6.7 Experiment 4 : Varying Bandwidth Availability and Constant Workload

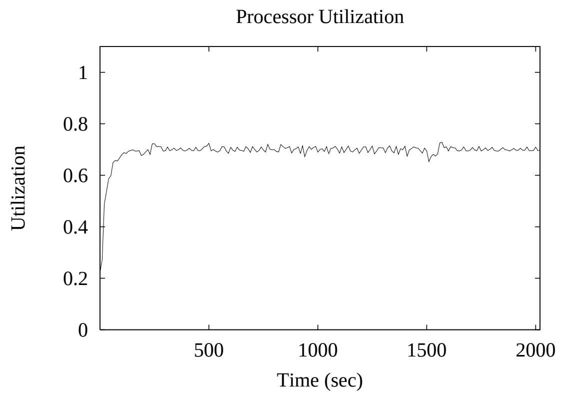
We now present the results obtained from running the experiment under varying channel capacity of the wireless network and a constant 2 number of objects of interest tracked by the system. This experiment demonstrates the adaptive resource management capabilities of HiDRA under varying resource availability and constant input workload. We normalize the channel capacity, bandwidth utilization, and bandwidth utilization set-point to the maximum channel capacity of 2Mbps. Table 8 summarizes the variation in channel capacity and bandwidth utilization set-point as a function of time. As it can be seen in Table 8, the

Time (sec)	Channel Capacity (Mbps)	Bandwidth Utilization Set-Point (Mbps)	Normalized Channel Capacity	Normalized Bandwidth Utilization Set-point
0 - 480	2.0	0.7 * 2.0 = 1.4	2.0 / 2.0 = 1.0	1.4 / 2.0 = 0.7
480 - 1,480	1.0	0.7 * 1.0 = 0.7	1.0 / 2.0 = 0.5	0.7 / 2.0 = 0.35
1,480 - 2,000	2.0	0.7 * 2.0 = 1.4	2.0 / 2.0 = 1.0	1.4 / 2.0 = 0.7

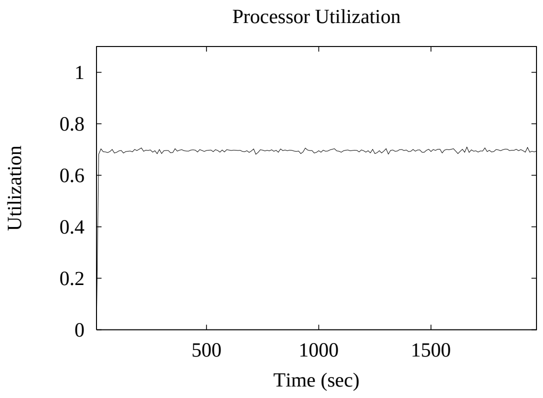
Table 8: Channel Capacity and Bandwidth Utilization Set-Point as a Function of Time

variation in the channel capacity represents a “step function”. A step function is selected because it is one of the most severe form of variation (or disturbance) that a control system can be subjected to. This experiment validates that HiDRA can maintain system stability even under such severe variation in channel capacity. Images containing objects of interests were captured by UAVs 1 and 2. In this experiment, the static configuration of the system shown in Table 3 was used when the system was operated without HiDRA.

II.6.7.1 Comparison of Resource Utilization



(a) Processor Utilization with HiDRA

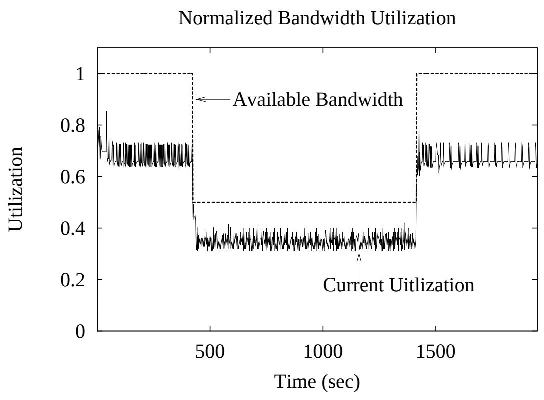


(b) Processor Utilization without HiDRA

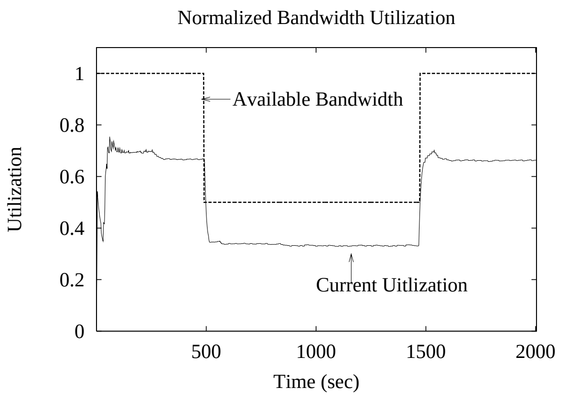
Figure 21: Exp 4: Comparison of Processor Utilization

Figures 21 and 22 compare the processor utilization at the receiver node and the normalized bandwidth utilization when the system was operated with and without HiDRA. The output of the bandwidth utilization monitor, shown in Figure 22b, was processed with a Kalman filter and used by the bandwidth controller as the current bandwidth utilization. From Figures 21a and 21b it can be seen that under this experimental scenario, processor utilization is equal to the set-point of 0.7 when the system was operated both with and without HiDRA.

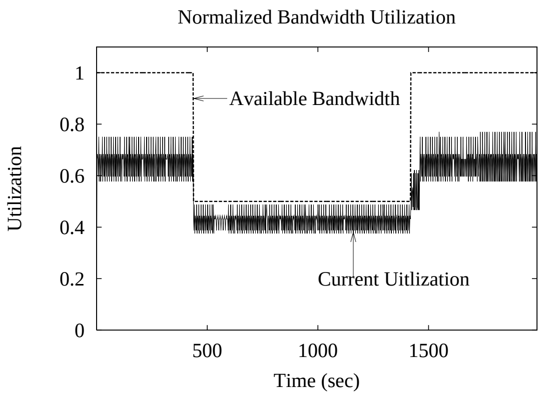
Figure 22c shows that when the system was operated without HiDRA, the normalized bandwidth utilization during $T \in [0s, 480s]$ and $T \in [1480s, 2000s]$ was 0.7, which is equal to the set-point. During $T \in [480s, 1480s]$ the normalized bandwidth utilization was 0.5, which is equal to the normalized channel capacity and significantly greater than the normalized set-point of 0.35. From Figures 22a and 22b, however, it can be seen that when the



(a) Normalized Bandwidth Utilization with HiDRA



(b) Normalized Bandwidth Utilization with HiDRA (processed using a Kalman Filter)



(c) Normalized Bandwidth Utilization without HiDRA

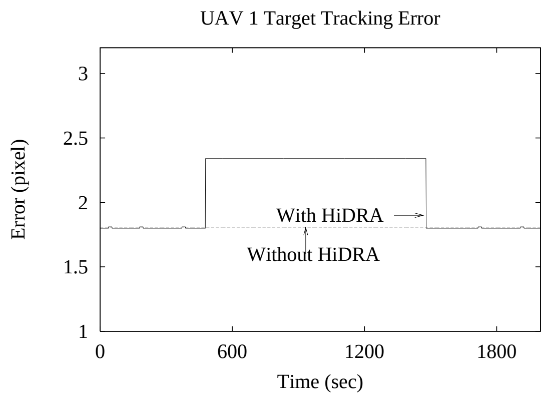
Figure 22: Exp 4: Comparison of Normalized Bandwidth Utilization

system was operated with HiDRA, the normalized bandwidth utilization converged to the normalized utilization set-point even under varying channel capacity. HiDRA achieved this behavior by lowering the quality factor of the images in response to fluctuations in network bandwidth.

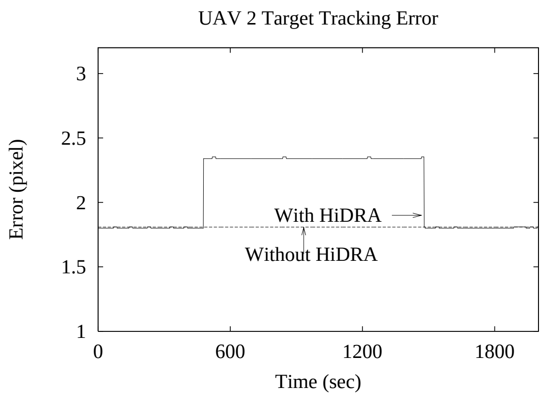
These results show that HiDRA ensures the wireless bandwidth utilization converges to the desired set-point within bounded time, even under varying network bandwidth availability. We therefore conclude that HiDRA ensures system resource utilization is maintained within the specified bounds, thereby ensuring system stability.

II.6.7.2 Comparison of QoS

We now compare the application QoS, which includes (1) target-tracking precision and (2) average end-to-end delay.



(a) UAV-1



(b) UAV-2

Figure 23: Exp 4: Comparison of Target-tracking Error

Figure 23 compares the target-tracking error that was obtained when the system operated with and without HiDRA. As shown in Table 8, during $T \in [0s, 480s]$ and $T \in [1,480s, 2000s]$ the channel capacity of the wireless network was 2 Mbps, which is the resource availability under which the static configuration of the system was selected. As a result, Figures 23a and 23b show that the target tracking error during during $T \in [0s, 480s]$ and $T \in [1,480s, 2000s]$ is nearly the same when the system was operated with HiDRA and without HiDRA.

During $T \in [480s, 1480s]$, however, the channel capacity of the wireless network was 1 Mbps. Within this time interval, the wireless bandwidth resource availability is half the wireless bandwidth resource availability under which the static configuration of the system was selected. To maintain the bandwidth utilization within specified bounds, HiDRA lowers the quality factor of the images transmitted by the UAVs to the receiver during

$T \in [480s, 1480s]$. As a result, Figures 23a and 23b show that the target tracking error during $T \in [480s, 1480s]$ was higher when the system was operated with HiDRA than when the system was operated without it. These results demonstrate that HiDRA effectively maintains utilization of system resource below the specified set-points despite variations in bandwidth resource availability by gracefully adjusting application QoS.

Table 9 compares the end-to-end delay when the system was operated with and without HiDRA. This table shows that end-to-end delay was much lower when the system was operated with HiDRA than without it. When the system was operated without HiDRA, during $T \in [480s, 1480s]$, the utilization of the wireless network bandwidth is equal to its channel capacity, which increased packet loss, retransmission delays, and in turn network transmission delay. This behavior accounts for the increase in the average end-to-end delay because the static configuration of the system was selected assuming 2 objects of interest were being tracked by the system and a constant channel capacity of 2 Mbps. When the system was operated with HiDRA, however, HiDRA reacts to variations in channel capacity by modifying application parameters such as JPEG quality factor. These adaptations ensure that system resources are not over-utilized and thus lowers average end-to-end delay.

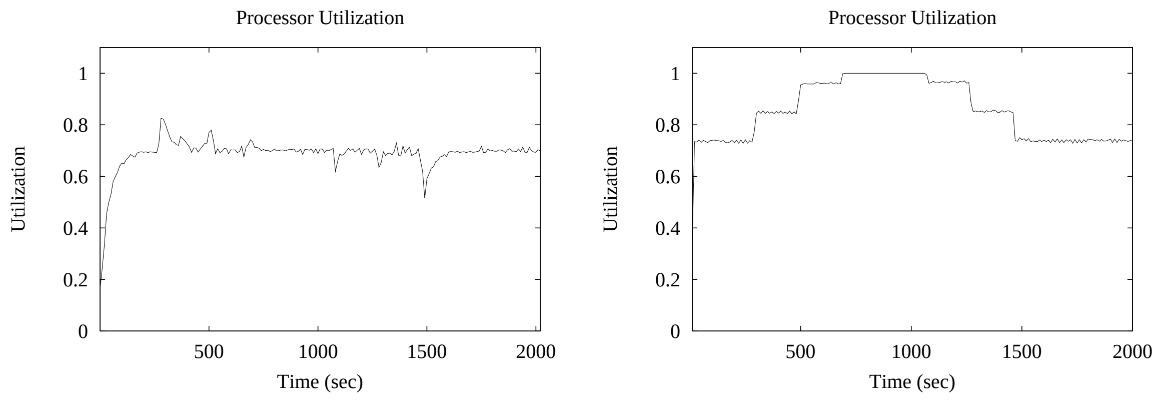
Number of Objects	End-to-End Delay (msec)	
	With HiDRA	Without HiDRA
2	185	276

Table 9: Exp 4: Comparison of End-to-End Delay

II.6.8 Experiment 5: Varying Bandwidth Availability and Varying Workload

We finally present the results obtained from running the experiment under varying channel capacity of the wireless network, as well as varying number of objects of interest in the system. This experiment demonstrates the adaptive resource management capabilities of HiDRA under varying resource availability and fluctuating input workload. We, once again, normalize the channel capacity, bandwidth utilization, and bandwidth utilization set-point to the maximum channel capacity of 2Mbps. Table 5 summarizes the number of objects of interests that were tracked as a function of time. Table 8 summarizes the variation of channel capacity as a function of time. In this experiment, when the system was operated without HiDRA, the static configuration of the system shown in Table 3 was used.

II.6.8.1 Comparison of Resource Utilization

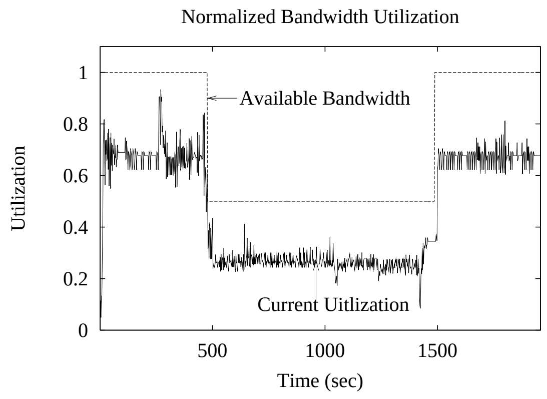


(a) Processor Utilization with HiDRA (b) Processor Utilization without HiDRA

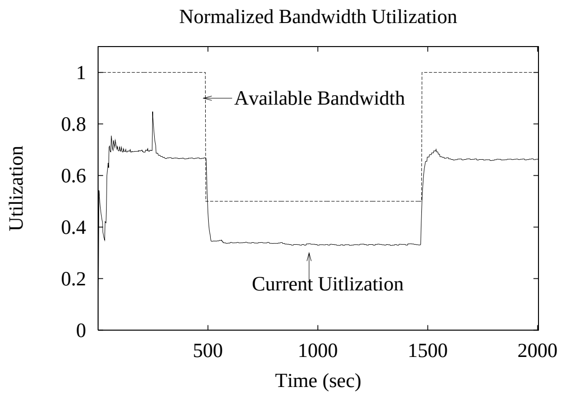
Figure 24: Exp 5: Comparison of Processor Utilization

Figures 24 and 25 compare the processor utilization at the receiver node and the wireless network bandwidth utilization when the system was operated with and without HiDRA. The output of the bandwidth utilization monitor, shown in Figure 25b, was processed with a Kalman filter and used by the bandwidth controller as the current bandwidth utilization.

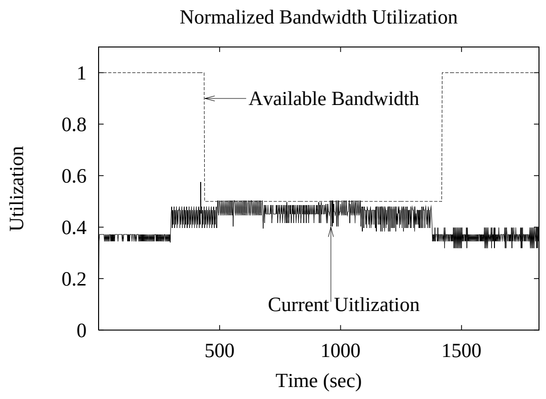
Figure 24 and Table 5 show that the increase in the processor utilization at $T = 300s$ is due to the presence of the first object of interest. From Figure 24a it can be seen that although the processor utilization increased above 0.7, within the next several sampling periods, HiDRA restored the processor utilization to the desired set-point of 0.7. This behavior was achieved by reducing the execution rates of data-source/receiver pair(s) deemed



(a) Normalized Bandwidth Utilization with HiDRA



(b) Normalized Bandwidth Utilization with HiDRA (processed using a Kalman Filter)



(c) Normalized Bandwidth Utilization without HiDRA

Figure 25: Exp 5: Comparison of Normalized Bandwidth Utilization

less important, *i.e.*, ones that captured images where objects of interest were absent. As shown in Figure 24b, when the system was operated without HiDRA, the processor utilization remained at 0.85, which is significantly higher than the utilization set-point of 0.7.

At $T = 500s$, the presence of the second object of interest was detected. As a result, Figure 24b shows that processor utilization increased to 0.95 when the system was operated without HiDRA. As shown in Figure 24a, however, the processor utilization quickly reconverges to the set-point after a transient increase when the system was operated with HiDRA.

At $T = 700s$ the presence of the third object of interest was detected. When the system was operated without HiDRA, Figure 24b shows how the processor utilization increased to 1. As shown in Figure 24a, however, once again the processor utilization quickly re-converges to the set-point after a transient increase when the system was operated with HiDRA.

At $T = 1100s$ the total number of objects currently being tracked by the system reduced from 3 to 2, although there was a decrease in the processor utilization, HiDRA restored the processor utilization of 0.7 by increasing the execution rate of important data-source/data sink pair(s). Similarly, HiDRA ensured that the processor utilization converges to the desired set-point for the remaining duration of the experiment.

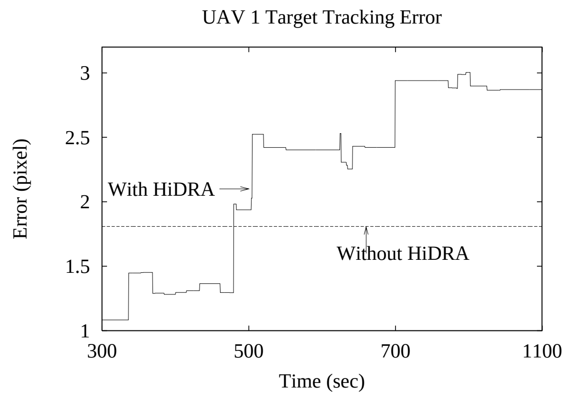
These results show that HiDRA ensures that the processor utilization of the receiver node converges to the desired set-point within bounded time, even under fluctuating workloads.

Figure 25c shows that when the system was operated without HiDRA, the normalized bandwidth utilization during $T \in [0s, 480s]$ and $T \in [1480s, 2000s]$ was below the normalized set-point of 0.7. During $T \in [480s, 1480s]$ the normalized bandwidth utilization was 0.5, which is equal to the channel capacity and significantly greater than the normalized set-point of 0.35. From Figures 25a and 25b, however, it can be seen that when the system operated with HiDRA, the normalized bandwidth utilization converged to the normalized utilization set-point even under varying channel capacity. This behavior was achieved by lowering the quality factor of the images in response to the variations in network bandwidth availability and input workload.

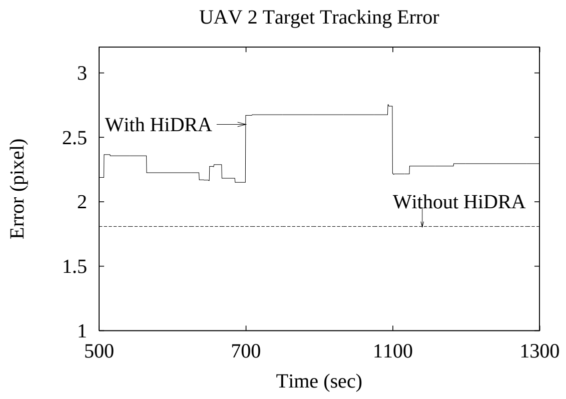
These results show that HiDRA ensures that the wireless bandwidth utilization converges to the desired set-point within bounded time, even under varying channel capacity and input workload. We therefore conclude that HiDRA ensures utilization of system resources is maintained within the specified bounds, even under varying resource availability and input workload, thereby ensuring system stability.

II.6.8.2 Comparison of QoS

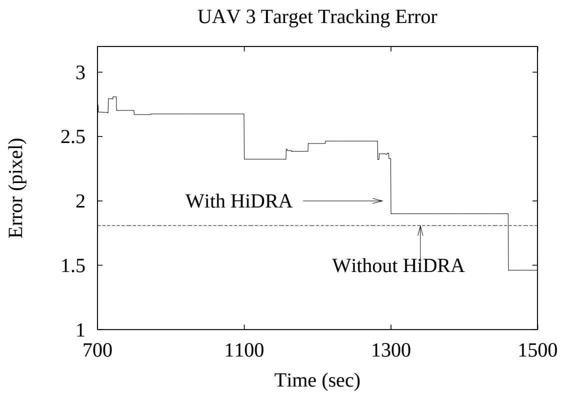
We now compare the application QoS – (1) target-tracking precision, and (2) average end-to-end delay.



(a) UAV-1



(b) UAV-2



(c) UAV-3

Figure 26: Exp 5: Comparison of Target-tracking Error

Figure 26 compares the target-tracking error that were obtained when the system was operated with and without HiDRA. Table 5 shows that during $T \in [300s, 500s]$ there was only one object of interest tracked by the system using UAV 1. When the system was operated without HiDRA, the static configuration of the system (as shown in Table 3) assumed (1) that there were a total of 2 objects of interests being tracked by the system and (2) a constant channel capacity of 2 Mbps. As a result, Figure 26a shows that the target tracking error during $T \in [300s, 480s]$ is lower when the system was operated with HiDRA than without it.

During $T \in [480s, 1480s]$, however, the channel capacity of the wireless network was 1 Mbps. During Within this time interval, the wireless network bandwidth availability was half the bandwidth availability under which the static configuration of the system was selected. To maintain the bandwidth utilization within specified bounds, therefore, HiDRA lowers the quality factor of the images transmitted by the UAVs to the receiver during $T \in [480s, 1480s]$. As a result, Figures 26a, 26b, and 26c show that the target tracking error during $T \in [480s, 1480s]$ was higher when the system was operated with HiDRA than without it.

These results demonstrate that HiDRA effectively maintains utilization of system resource below the specified set-points despite fluctuations in input workload and variations in bandwidth resource availability by gracefully adjusting application QoS.

Table 10 compares the end-to-end delay when the system was operated with and without HiDRA. This table shows that end-to-end delay is much lower when the system operates

Number of Objects	End-to-End Delay (msec)	
	With HiDRA	Without HiDRA
0	20	20
1	80	123
2	137	235
3	206	327

Table 10: Exp 5: Comparison of End-to-End Delay

with HiDRA than without it. When the system was operated without HiDRA the utilization of the wireless network bandwidth is equal to its channel capacity during $T \in [480s, 1480s]$, which resulted in increased packet loss, retransmission delays, which in turn increased network transmission delay. This behavior accounts for the increase in the average end-to-end delay because the static configuration of the system was selected assuming 2 objects of interest were being tracked by the system and a constant channel capacity of 2 Mbps. When the system was operated with HiDRA, however, it reacts to variations in channel capacity and number of objects by modifying application parameters, such as the JPEG quality factor. These adaptations ensures that system resources are not over-utilized and thus lowers average end-to-end delay.

II.6.9 Summary

HiDRA responds to fluctuation in input workload and the most severe form of variation in resource availability by periodically monitoring and control of resource utilization. Both our theoretical and empirical analysis assures that the utilization of system resources converge to their specified utilization set-points even if a set-point is specified as a time-varying reference signal. However, the only assumption is that the variation in the reference signal is slower than the sampling period.

Our results show that when resources utilization increases above the desired set-point, HiDRA lowers the utilization by modifying application parameters, such as execution rates and JPEG quality factor. These adaptations ensure that (1) system resources are not over-utilized and (2) enough resources are available for important applications. Our results also show that when the system was operated with independent feedback loops, system resources are severely under-utilized, and as a result application QoS are significantly reduced.

Our analysis of the results described above suggests that applying hierarchical adaptive resource management to our target tracking system helps to (1) maintain system resource utilization within specified bounds and (2) improve overall system QoS. These improvements are achieved largely due to monitoring of system resource utilization, adaptive resource provisioning, and efficient system workload management by means of HiDRA's resource monitors, hierarchical controllers, and effectors, respectively.

II.7 Summary

This chapter described HiDRA, which is our hierarchical distributed resource management architecture based on control-theoretic techniques that provides adaptive resource management, such as resource monitoring and application adaptation, that are key to supporting open DRE systems. We first presented the theoretical analysis that shows how HiDRA ensures stability in our DRE system. We then evaluated the performance of HiDRA using a representative target tracking DRE system implemented using Real-time CORBA and composed of two types of system resources (*i.e.*, computational power at the receiver and wireless network bandwidth) and three applications (*i.e.*, UAV data sender/receiver pairs). Our theoretical analysis and empirical results show that HiDRA delivers efficient resource utilization by maintaining system resource utilization within specified bounds even under fluctuating work loads, thereby ensuring system stability and delivering effective QoS. However, as HiDRA tries to achieve the desired utilization set-point of system resources at all times, where there is no resource contention between applications executing in the system, the system can be operated without HiDRA to conserve system resources. When resource contention arises, the system can be operated with HiDRA to ensure that the utilization of system resources is maintained within the specified set-point.

CHAPTER III

ADAPTIVE RESOURCE MANAGEMENT FRAMEWORKS

Achieving end-to-end quality of service (QoS) in DRE systems requires integrating a range of real-time capabilities, such as QoS-enabled network protocols, real-time operating system scheduling mechanisms and policies, and real-time middleware services, across the system domain. Although existing research and solutions [13, 54] focus on improving the performance and QoS of individual capabilities of the system (such as operating system scheduling mechanism and policies), they are not sufficient for DRE systems as these systems require integrating a range of real-time capabilities across the system domain. Conventional QoS-enabled middleware technologies, such as Real-time CORBA [62] and the Real-time Java [10], have been used extensively as an operating platforms to build DRE systems as they support explicit configuration of QoS aspects (such as priority and threading models), and provide many desirable real-time features (such as priority propagation, scheduling services, and explicit binding of network connections).

QoS-enabled middleware technologies have traditionally focused on DRE systems that operate in *closed* environments where operating conditions, input workloads, and resource availability are known in advance and do not vary significantly at runtime. An example of a closed DRE system is an avionics mission computer [72], where the penalty of not meeting a QoS requirement (such as deadline) can result in the failure of the entire system or mission. Conventional QoS-enabled middleware technologies are insufficient, however, for DRE systems that execute in *open* environments where operational conditions, input workload, and resource availability cannot be characterized accurately *a priori*. Examples of open DRE systems include shipboard computing environments [68], multi-satellite missions [78], and intelligence, surveillance and reconnaissance missions [71].

Specifying and enforcing end-to-end QoS is an important and challenging issue for open systems DRE due to their unique characteristics, including (1) constraints in multiple resources (*e.g.*, limited computing power and network bandwidth) and (2) highly fluctuating resource availability and input workload. At the heart of achieving end-to-end QoS are resource management techniques that enable open DRE systems to *adapt* to dynamic changes in resource availability and demand. In earlier work we developed adaptive resource management *algorithms* (such as EUCON [52], DEUCON [83], HySUCON [41], and FMUF [18]) and *architectures*, such as HiDRA [70] based on control-theoretic techniques. We then developed FC-ORB [84], which is a QoS-enabled adaptive middleware that implements the EUCON algorithm to handle fluctuations in application workload and system resource availability.

A limitation with our prior work, however, is that it tightly coupled resource management algorithms within particular middleware platforms, which made it hard to enhance the algorithms without redeveloping significant portions of the middleware. For example, since the design and implementation of FC-ORB was closely tied to the EUCON adaptive resource management algorithm, significant modifications to the middleware was needed to support other resource management algorithms, such as DEUCON, HySUCON, or FMUF. Object-oriented frameworks have traditionally been used to factor out many reusable general-purpose and domain-specific services from DRE systems and applications [66]; however, to alleviate the tight coupling between resource management algorithms and middleware platforms and improve flexibility, this paper presents a *adaptive resource management framework* for open DRE systems. Contributions of this chapter to the study of adaptive resource management solutions for open DRE systems include:

• **The design of a Resource Allocation and Control Engine (RACE)**, which is a fully customizable and configurable adaptive resource management framework for open DRE systems. RACE decouples adaptive resource management algorithms from the middleware implementation, thereby enabling the usage of various resource management algorithms without the need for redeveloping significant portions of the middleware. RACE can be configured to support a range of algorithms for adaptive resource management without requiring modifications to the underlying middleware. To enabling the seamless integration of resource allocation and control algorithms into DRE systems, RACE enables the deployment and configuration of feedback control loops. RACE therefore complements theoretical research on adaptive resource management algorithms that provide a model and theoretical analysis of system performance.

As shown in Figure 27, RACE provides (1) *resource monitors* that track utilization of various system resources, such as CPU, memory, and network bandwidth, (2) *QoS monitors* that track application QoS, such as end-to-end delay, (3) *resource allocators* that allocate resource to components based on their resource requirements and current availability of system resources, (4) *configurators* that configure middleware QoS parameters of application components, (5) *controllers* that compute end-to-end adaptation decisions based on control algorithms to ensure that QoS requirements of applications are met, and (6) *effectors* that perform controller-recommended adaptations.

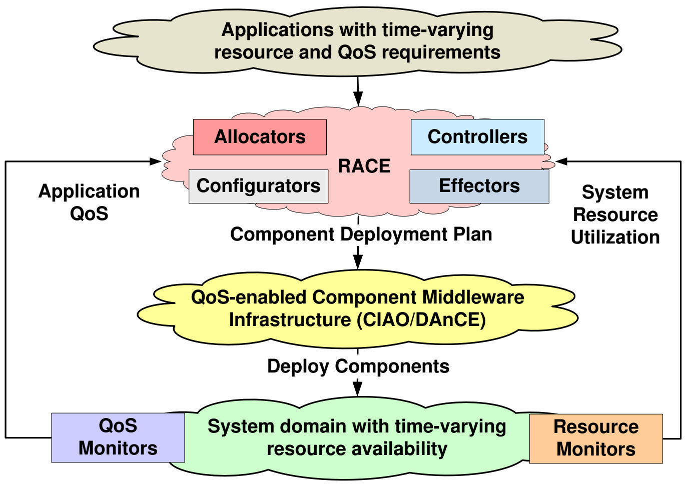

Applications with time-varying resource and QoS requirements
Allocators
Controllers
RACE
Configurators
Effectors
Application QoS
System Resource Utilization
Component Deployment Plan
QoS-enabled Component Middleware Infrastructure (CIAO/DAnCE)
Deploy Components
QoS Monitors
System domain with time-varying resource availability
Resource Monitors

Figure 27: A Resource Allocation and Control Engine (RACE) for Open DRE Systems

• **The empirical evaluation of RACE's scalability** as the number of nodes and applications in a DRE system grows. Scalability is an integral property of a framework as it determines the framework's applicability. Since open DRE systems comprise large number of nodes and applications, to determine whether RACE can be applied to such systems, we empirically evaluate RACE's scalability as the number of applications and nodes in the system increases. Our results demonstrate that RACE scales well as the number of applications and nodes in the system increases, and therefore can be applied to a wide range of open DRE systems.

The remainder of the chapter is organized as follows: Section III.1 compares our research on RACE with related work; Section III.2 describes the architecture of RACE; Section III.3 presents an empirical measure of RACE's scalability as the number of applications and nodes in the system grows; and Section III.4 concludes the chapter by presenting a summary.

III.1 Related Research

This section presents an overview of existing middleware technologies that have been used to develop open DRE system. As in Figure 28 and described below, we classify this research along two orthogonal dimensions: (1) QoS-enabled DOC middleware vs. QoS-enabled component middleware and (2) design-time vs. run-time QoS configuration, optimization, analysis, and evaluation of constraints, such as timing, memory, and CPU.

III.1.1 Conventional and QoS-enabled DOC Middleware

Conventional middleware technologies for distributed object computing (DOC), such as The Object Management Group (OMG)'s CORBA [59] and Sun's Java RMI [76], encapsulates and enhances native OS mechanisms to create reusable network programming components. These technologies provide a layer of abstraction that shields application developers from the low-level platform-specific details and define higher-level distributed programming models whose reusable APIs and components automate and extend native OS capabilities.

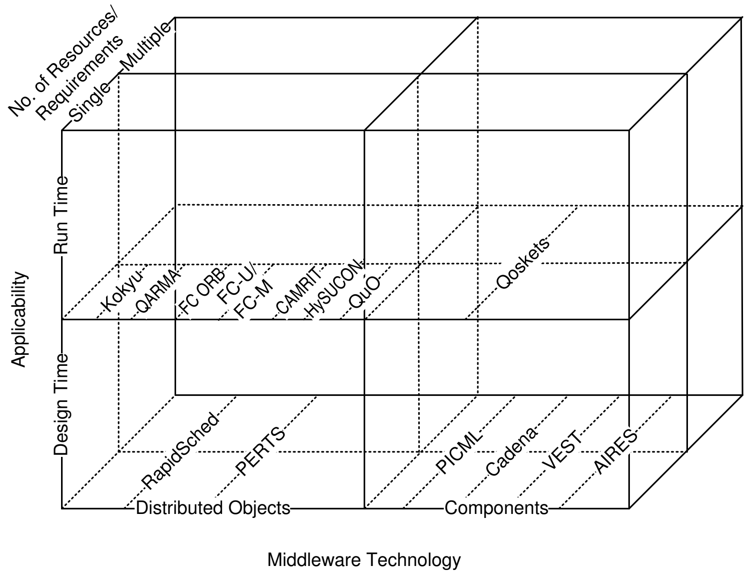


Figure 28: Taxonomy of Related Research

Conventional DOC middleware technologies, however, address only *functional* aspects of system/application development such as how to define and integrate object interfaces and implementations. They do not address QoS aspects of system/application development such as how to (1) define and enforce application timing requirements, (2) allocate resources to applications, and (3) configure OS and network QoS policies such as priorities for application processes and/or threads. As a result, the code that configures and manages QoS aspects often become entangled with the application code. These limitations with conventional DOC middleware have been addressed by the following run-time platforms and design-time tools:

Run-time. Early work on resource management middleware for shipboard DRE systems presented in [64, 86] motivated the need for adaptive resource management middleware. This work was further extended by QARMA [30], which provides resource management as a *service* for existing QoS-enabled DOC middleware, such as RT-CORBA. Kokyu [33] also enhances RT-CORBA QoS-enabled DOC middleware by providing a portable middleware scheduling framework that offers flexible scheduling and dispatching services. Kokyu performs feasibility analysis based on estimated worst case execution times of applications to determine if a set of applications is *schedulable*. Resource requirements of applications, such as memory and network bandwidth, are not captured and taken into consideration by Kokyu. Moreover, Kokyu lacks the capability to track utilization of various system resources as well as QoS of applications. To address these limitations, research presented in [15] enhances QoS-enabled DOC middleware by combining Kokyu and QARMA.

Design-time. RapidSched [88] enhances QoS-enabled DOC middleware, such as RT-CORBA, by computing and enforcing distributed priorities. RapidSched uses PERTS [47] to specify real-time information, such as deadline, estimated execution times, and resource requirements. Static schedulability analysis (such as rate-monotonic analysis) is then performed and priorities are computed for each CORBA object in the system. After the priorities are computed, RapidSched uses RT-CORBA features to enforce these computed priorities.

III.1.2 Conventional and QoS-enabled Component Middleware

Conventional component middleware technologies, such as the CORBA Component Model (CCM) [60] and Enterprise Java Beans [5, 77], provide capabilities that addresses the limitation of DOC middleware technologies in the context of system design and development. Examples of additional capabilities offered by conventional component middleware compared to conventional DOC middleware technology include (1) standardized interfaces for application component interaction, (2) model-based tools for deploying and interconnecting components, and (3) standards-based mechanisms for installing, initializing, and configuring application components, thus separating concerns of application development, configuration, and deployment.

Although conventional component middleware support the design and development of large scale distributed systems, they do not address the address the QoS limitations of DOC middleware. Therefore, conventional component middleware can support large scale enterprise distributed systems, but not DRE systems that have the stringent QoS requirements.

These limitations with conventional component-based middleware have been addressed by the following run-time platforms and design-time tools:

Run-time. QoS provisioning frameworks, such as QuO [90] and Qoskets [53, 65, 71] help ensure desired performance of DRE systems built atop QoS-enabled DOC middleware and QoS-enabled component middleware, respectively. When applications are designed using Qoskets (1) resources are dynamically (re)allocated to applications in response to changing operational conditions and/or input workload and (2) application parameters are fine-tuned to ensure that allocated resource are used effectively. With this approach, however, applications are augmented explicitly at design-time with Qosket components, such as monitors, controllers, and effectors. This approach thus requires redesign and reassembly of existing applications built without Qoskets. When applications are generated at run-time (*e.g.*, by intelligent mission planners [39]), this approach would require planners to augment the applications with Qosket components, which may be infeasible since planners are designed and built to solve mission goals and not perform such platform-/middleware-specific operations.

Design-time. VEST [74] is a design assistant tool based on the *Generic Modeling Environment* [4] that enables embedded system composition from component libraries and checks whether timing, memory, power, and cost constraints of real-time and embedded applications are satisfied. AIRES [40] is a similar tool that provides the means to map design-time models of component composition with real-time requirements to run-time models that weave together timing and scheduling attributes. The research presented in [48] describes a design assistant tool, based on MAST [34], that comprises a DSML and a suite of analysis and system QoS configuration tools and enables composition, schedulability analysis, and assignment of operating system priority for application components. Cadena [35] is an integrated environment for developing and verifying component-based DRE systems by applying static analysis, model-checking, and lightweight formal methods. Cadena also provides a component assembly framework for visualizing and developing components and their connections.

Some design-time tools, such as AIRES, VEST, and those presented in [48], use *estimates*, such as estimated worst case execution time, estimated CPU, memory, and/or network bandwidth requirements. These tools are targeted for systems that execute in *closed* environments, where operational conditions, input workload, and resource availability can be characterized accurately *a priori*. Since RACE tracks and manages utilization of various system resources, as well as application QoS, it can be used in conjunction with these tools to build open DRE systems.

III.1.3 Unresolved Challenges

We now describe the shortcomings of existing research on resource management tools and frameworks for large-scale DRE systems, focusing on the key research challenges that are still unresolved.

Design-time solutions. As described earlier in Section II.3, design time solutions – for both DOC middleware and component middleware – perform analysis and resource management using *estimates*, such as estimated worst case execution time, estimated CPU, memory, and/or network bandwidth requirements. These tools and techniques are targeted for systems that execute in *closed* environments, where operational conditions, input workload, and resource availability can be characterized accurately *a priori*. What is needed is a resource management framework that tracks and manages utilization of various system resources, as well as application QoS, that can be used in conjunction with these tools to build DRE systems that execute in open environments.

Runtime solutions for QoS-enabled DOC middleware. As these solutions are built atop DOC middleware, they inherit the limitations of DOC middleware described in I.1. As a result, these solutions do not provide higher level abstraction that separates the framework configuration from framework functionality. Configuration and customization of these frameworks are done via source code, and therefore is tedious and error-prone. With existing solutions, incorporation of new resource management algorithms into the resource management framework would involve reimplementation of significant portions of the middleware or framework. Moreover, existing solutions assume resources are already allocated to applications and do not perform on-line resource allocation or admission control.

Runtime solutions for QoS-enabled component middleware. With existing solution approaches, applications are augmented explicitly at design-time with Qosket components, such as monitors, controllers, and effectors. This approach thus requires redesign and reassembly of existing applications built without Qoskets, which might not be feasible for DRE systems with large number of legacy applications. Moreover, when applications are generated at run-time (*e.g.*, by intelligent mission planners [39]), this approach would require planners to augment the applications with Qosket components, which may be infeasible since planners are designed and built to solve mission goals and to work atop any component middleware, not just CCM.

In summary, what is missing is a resource management framework that provides adaptive resource and QoS management capabilities in an application transparent and non-intrusive way. In particular, the framework should allocate CPU, memory, and networking resources to application components and track and manage utilization of various system resources, as well as application QoS. The framework should have the capability to deploy and manage applications that are composed at design-time by system designers (using DSML tools such as PICML), as well as at run-time by intelligent mission planners. The framework should provide reusable entities, such as resource monitors, QoS monitors, and effectors, that can be configured to incorporate a range of existing control algorithms, such as EUCON [52] and HySUCON [41], as well as future algorithms. Moreover, the framework should provide higher level of abstractions that aid in configuring and customizing the framework.

III.2 Structure and Functionality of RACE

This section describes the structure and functionality of RACE. RACE supports open DRE systems built atop CIAO, which is an open-source implementation of Lightweight CCM. All entities of RACE themselves are designed and implemented as CCM components, so RACE's `Allocators` and `Controllers` can be configured to support a range of resource allocation and control algorithms using model-driven tools, such as PICML.

Figure 29 elaborates the earlier architectural overview of RACE in Figure 27 and shows how the detailed design of RACE is composed of the following components: (1)

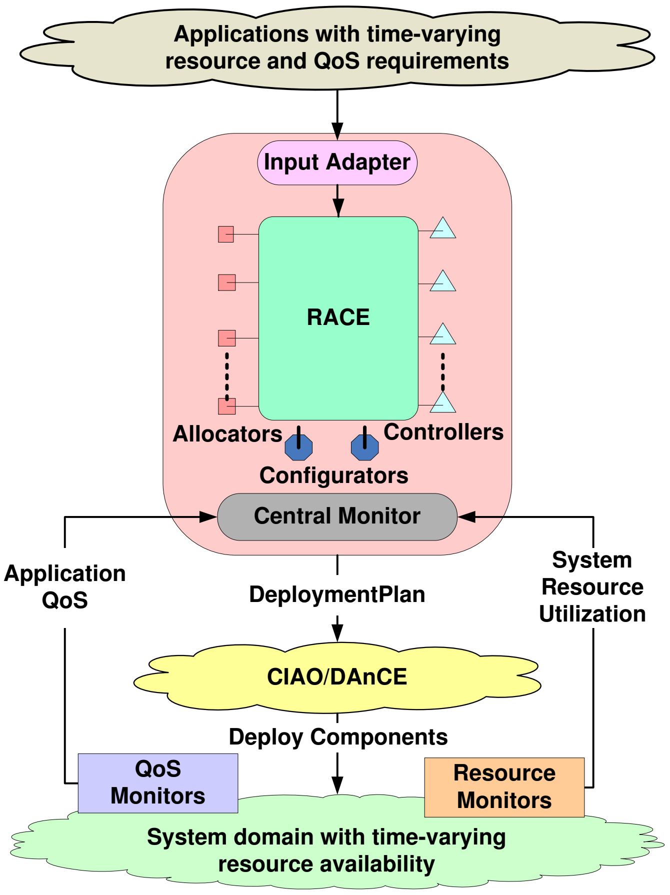


Figure 29: Detailed Design of RACE

`InputAdapter`, (2)`CentralMonitor` , (3) `Allocators`, (4) `Configurators`,

(5) `Controllers`, and (6) `Effectors`. RACE monitors application QoS and system resource usage via its `Resource Monitors`, `QoS-Monitors`, `Node Monitors` and `CentralMonitor`. Each component in RACE is described below in the context of the overall adaptive resource management challenge it addresses.

III.2.0.1 Challenge 1: Configuration and Customization of the Resource Management Framework

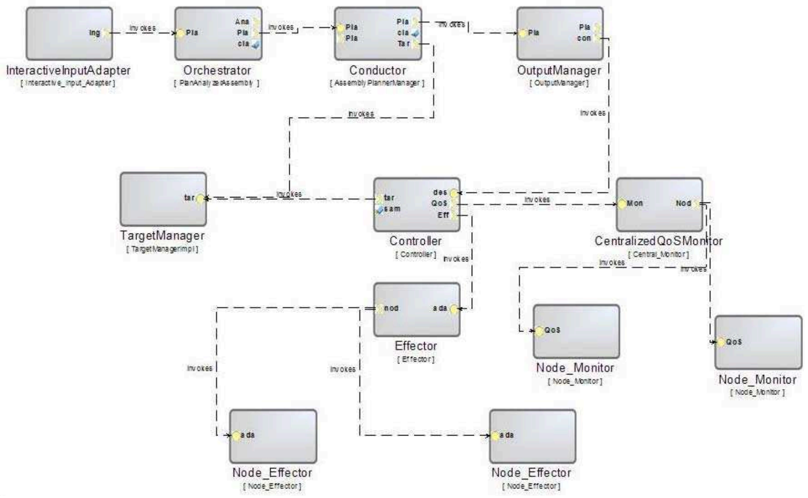


Figure 30: PICML Model of RACE

Problem. Configuration and customization of existing resource management frameworks described in Section III.1 are done via source code, and therefore is tedious and error-prone. With existing resource management frameworks, incorporation of new resource management algorithms into the resource management framework would involve reimplementation of significant portions of the middleware or framework.

Solution. RACE's novelty stems from its combination of design-time DSML tools and QoS-enabled component middleware run-time platforms. RACE's reusable entities, such as resource monitors, QoS monitors, implementation of resource management algorithms, and effectors, can be configured to incorporate a range of existing control algorithms, such as EUCON and HySUCON, as well as future algorithms. The elements of RACE are designed and implemented as CCM components, and therefore as shown in Figure 30, RACE can be configured using DSML tools such as PICML. RACE provides a higher level of abstraction to configure/customize the framework compared to other existing resource management frameworks, which are configured via source code.

Since system QoS monitors and effectors tend to be domain specific, RACE provides the capability to "plug-in" domain specific entities. Moreover, as implementation of resource management algorithms in RACE are encapsulated as components, RACE separates the concerns of resource management algorithms and the middleware.

III.2.0.2 Challenge 2: Domain Specific Representation of Application Metadata

Problem. End-to-end applications can be composed either at design time or at runtime. At design time, CCM based end-to-end applications are composed using model-driven tools, such as PICML; and at runtime, they can be composed by intelligent mission planners like such as the *spreading activation partial order planner* (SA-POP) [39]. When an application is composed using PICML, metadata describing the application is captured in XML files based on the `PackageConfiguration` schema defined by the Object Management Group's Deployment and Configuration specification [61]. When applications are generated during runtime by SA-POP, metadata is captured in an in-memory structure defined by the planner.

Solution: Domain-specific customization and configuration of RACE's adapters. During design time, RACE can be configured using PICML and an `InputAdapter` appropriate for the domain/system can be selected. For example, to manage a system in which applications are constructed at design-time using PICML, RACE can be configured with the `PICMLInputAdapter`; and to manage a system in which applications are constructed at runtime using SA-POP, RACE can be configured with the `SAPOPInputAdapter`. As

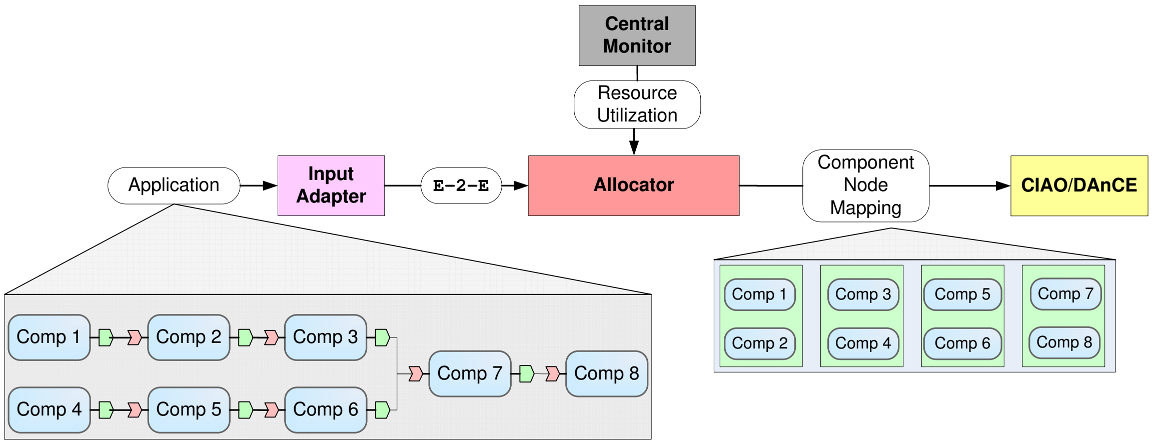


Figure 31: Resource Allocation to Application Components Using RACE

shown in Figure 31, the `InputAdapter` parses the metadata that describes the application into an in-memory end-to-end (`E-2-E`) IDL structure that is internal to RACE. Key entities of the `E-2-E` IDL structure are shown in Figure 32.

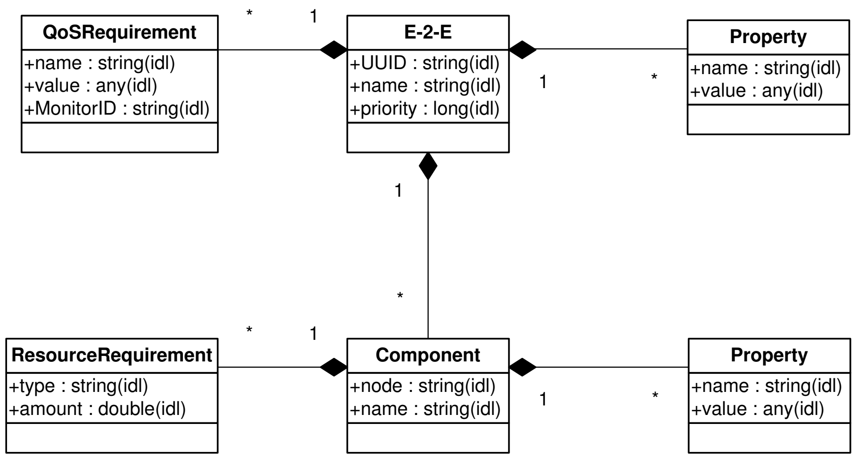


Figure 32: Main Entities of RACE's E-2-E IDL Structure

The `E-2-E` IDL structure populated by the `InputAdapter` contains information regarding the application, including (1) components that make up the application and their resource requirement(s), (2) interconnections between the components, (3) application QoS properties (such relative priority) and QoS requirement(s) (such as end-to-end delay), and

(4) mapping components onto domain nodes. The mapping of components onto nodes need not be specified in the metadata that describes the application which is given to RACE. If an mapping is specified, it is honored by RACE; if not, a mapping is determined at runtime by RACE's `Allocators`.

III.2.0.3 Challenge 3: Efficient Monitoring of System Resource Utilization and Application QoS

Problem. In open DRE systems, input workload, application QoS, and utilization and availability of system resource are subject to dynamic variations. In order to ensure application QoS requirements are met, as well as utilization of system resources are within specified bounds, application QoS and utilization/availability of system resources are to be monitored periodically. The key challenge lies in designing and implementing a resource and QoS monitoring architecture that scales well as the number of applications and nodes in the system increase.

Solution: Hierarchical QoS and resource monitoring architecture. RACE's monitoring framework is composed of the `Central Monitor`, `Node Monitors`, `Resource Monitors`, and `QoS Monitors`. These components track resource utilization by, and QoS of, application components. As shown in Figure 33, RACE's `Monitors` are structured in the following hierarchical fashion. A `Resource Monitor` collects resource utilization metrics of a specific resource, such as CPU or memory. A `QoS Monitor` collects specific QoS metrics of an application, such as end-to-end latency or throughput. A `Node Monitor` tracks the QoS of all the applications running on a node as well as the resource utilization of that node. Finally, a `Central Monitor` tracks the QoS of all the applications running the entire system, which captures the system QoS, as well as the resource utilization of the entire system, which captures the system resource utilization.

`Resource Monitors` use the operating system facilities, such as `/proc` file system in `Linux/Unix` operating systems and the *system registry* in `Windows` operating

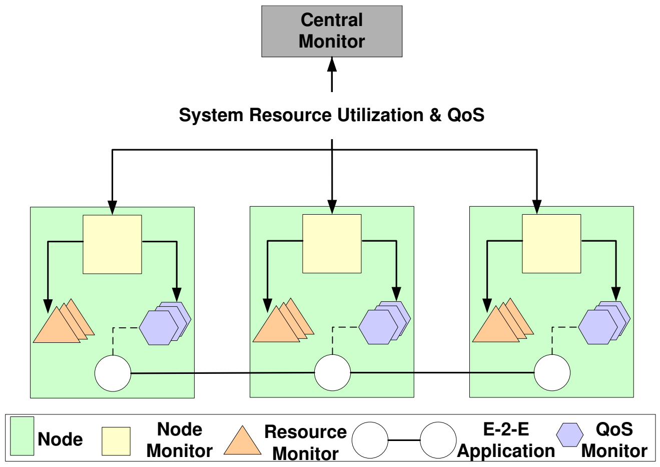


Figure 33: Architecture of Monitoring Framework

systems, to collect resource utilization metrics of that node. As the resource monitors are implemented as shared libraries that can be loaded at runtime, RACE can be configured with new/domain-specific resource monitors without making any modifications to other entities of RACE. `QoS Monitors` are implemented as software modules that collect end-to-end latency and throughput metrics of an application and are dynamically installed into a running system using DyInst [16]. This approach ensure rebuilding, re-implementation, or re-starting of already running application components is not required. Moreover, with this approach, `QoS Monitors` can be turned on or off on demand at runtime.

The primary metric that we use to measure the performance of our monitoring framework is *monitoring delay*, which is defined as the time taken to obtain a snapshot of the entire system in terms of resource utilization and QoS. To minimize the monitoring delay and ensure that RACE's monitoring architecture scales as the number of applications and nodes in the system increase, the RACE's monitoring architecture is structured in a hierarchical fashion. We validate this claim in Section III.3.

III.2.0.4 Challenge 4: Resource Allocation

Problem. Applications executing in open DRE systems are resource sensitive and require multiple resources such as memory, CPU, and network bandwidth. In open DRE systems, resources allocation cannot be performed during design time as system resource availability may be time variant. Moreover, input workload affects the utilization of system resources by already executing applications. Therefore, the key challenge lies in allocating various systems resources to application components in a timely fashion.

Solution:On-line Resource allocation. RACE's `Allocators` implement resource allocation algorithms and allocate various domain resources (such as CPU, memory, and network bandwidth) to application components by determining the mapping of components onto nodes in the system domain. For certain applications, *static* mapping between components and nodes may be specified at design-time by system developers. To honor these static mappings, RACE therefore provides a *static allocator* that ensures components are allocated to nodes in accordance with the static mapping specified in the application's metadata. If no static mapping is specified, however, *dynamic allocators* determine the component to node mapping at runtime based on resource requirements of the components and current resource availability on the various nodes in the domain. As shown in Figure 31, input to `Allocators` include the `E-2-E` IDL structure corresponding to the application and the current utilization of system resources.

The current version of RACE provides the following `Allocators`: (1) a single dimension bin-packer [42] that makes allocation decisions based on either CPU, memory, or network-bandwidth requirements and availability, (2) a multi-dimensional bin-packer – partitioned breadth first decreasing allocator [24] – that makes allocation decisions based on CPU, memory, and network-bandwidth requirements and availability, and (3) a static allocator. Metadata is associated with each allocator and captures its type (*i.e.*, static, single dimension bin-packing, or multi-dimensional bin-packer) and associated resource overhead (such as CPU and memory utilization). Since `Allocators` themselves are CCM components, RACE can be configured with new `Allocators` by using PICML.

III.2.0.5 Challenge 5: Accidental Complexities in Configuring Platform-specific QoS Parameters

Problem. As described in Section IV.2.2, real-time QoS configuration of the underlying component middleware, operating system, and network affects the QoS of applications executing in open DRE systems. Since these configurations are platform-specific, it is tedious and error-prone for system developers or SA-POP to specify them in isolation.

Solution: Automate configuration of platform-specific parameters. As shown in Figure 34, RACE's `Configurators` determine values for various low-level platform-specific QoS parameters, such as middleware, operating system, and network settings for an application based on its QoS characteristics and requirements such as relative importance and end-to-end delay. For example, the `MiddlewareConfigurator` configures com-

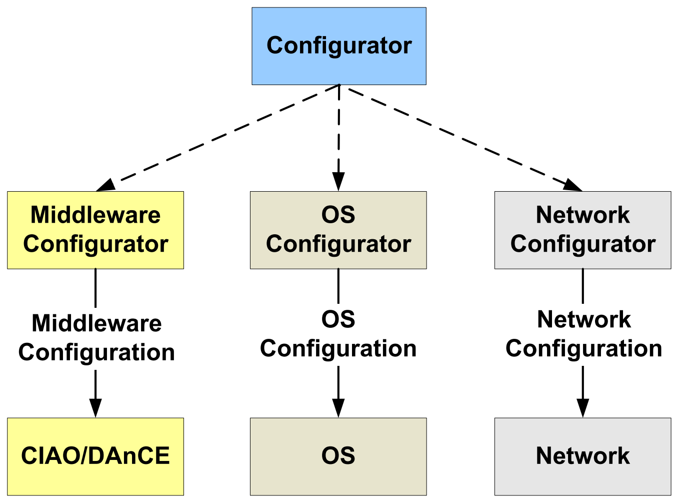


Figure 34: QoS Parameter Configuration with RACE

ponent Lightweight CCM policies, such as threading policy, priority model, and request processing policy based on the class of the application (*important* and *best-effort*). The `OperatingSystemConfigurator` configures operating system parameters, such as the priorities of the *Component Servers* that host the components based on Rate Monotonic Scheduling (RMS) [42] or based on criticality (relative importance) of the application. Likewise, the `NetworkConfigurator` configures network parameters, such as `diffserv` code-points of the component interconnections. Like other entities of RACE, `Configurators` are implemented as CCM components, so new configurators can be plugged into RACE by configuring RACE at design-time using PICML.

III.2.0.6 Challenge 6: Computation of System Adaptation Decisions

Problem. In open DRE systems, resource utilization of applications might be significantly different than their estimated values and availability of system resources may be time-variant. Moreover, for applications executing in these systems, the relation between input workload, resource utilization, and QoS cannot be characterized *a priori*. Therefore, in order to ensure that QoS requirements of applications are met, and utilization system resources are within the specified bounds, the system must be able to *adapt* to dynamic changes, such as variations in operational conditions, input workload, and/or resource availability.

Solution: Use of Control-theoretic adaptive resource management algorithms. RACE's `Controllers` implement various Control-theoretic adaptive resource management algorithms such as EUCON [52], DEUCON [83], HySUCON [41], and FMUF [18], thereby enabling open DRE systems to adapt to changing operational context and variations in resource availability and/or demand. Based on the control algorithm they implement, `Controllers` modify configurable system parameters, such as execution rates and mode of operation of the application, real-time configuration settings – operating system priorities of *component servers* that host the components – and network `difserv` code-points of the component interconnections. As shown in Figure 35, input to the controllers include current resource utilization and current QoS. Since `Controllers` are implemented as

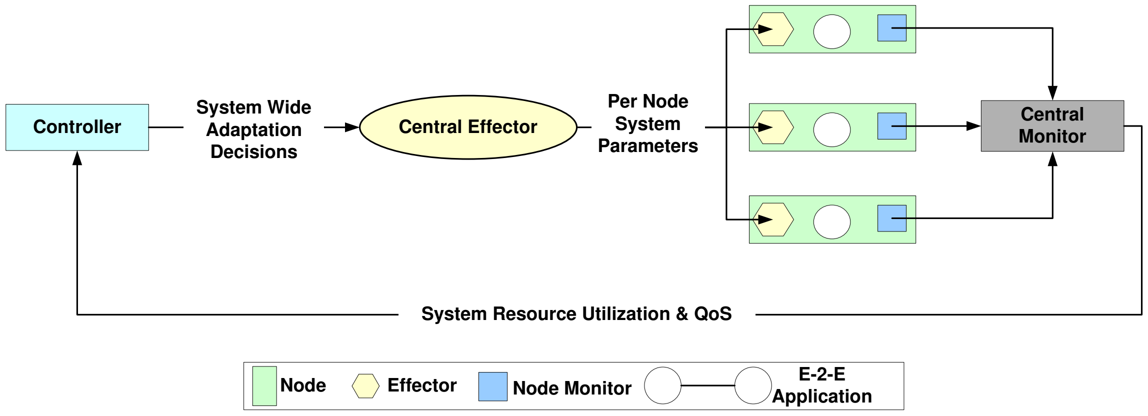

Controller
System Wide Adaptation Decisions
Central Effector
Per Node System Parameters
Central Monitor
System Resource Utilization & QoS
Node
Effector
Node Monitor
E-2-E Application

Figure 35: RACE's Feedback Control Loop

CCM components RACE can be configured with new `Controllers` by using PICML.

III.2.0.7 Challenge 7: Efficient Execution of System Adaptation Decisions

Problem. Although control-theoretic adaptive resource management algorithms compute system adaptation decisions, one of the challenges we faced in building RACE is the design and implementation of *effectors* – entities that modify system parameters in order to achieve the controller recommended system adaptation. The key challenge lies in designing and implementing the effector architecture that scales well as the number of applications and nodes in the system increase.

Solution: Hierarchical effector architecture. Effectors modify application parameters, including resources allocated to components, execution rates of applications, and system parameters including OS, middleware, and network QoS setting for components, to achieve the controller recommended adaptation. As shown in Figure 35, `Effectors` are designed hierarchically. The `Central Effector` first computes the values of various system parameters for all the nodes in the domain to achieve the `Controller` recommended adaptation. The computed values of system parameters for each node are then propagated to `Effectors` located on each node, which then modify system parameters of its node accordingly.

The primary metric that is used to measure the performance of a monitoring effectors is *actuation delay*, which is defined as the time taken to execute controller recommended adaptation throughout the system. To minimize the actuation delay and ensure that RACE scales as the number of applications and nodes in the system increase, the RACE's effectors are structured in a hierarchical fashion. We validate this claim in Section III.3.

Since the elements of RACE are developed as CCM components, RACE itself can be configured using model-driven tools, such as PICML. Moreover, new and/or domain specific entities such as`InputAdapters`, `Allocators`, `Controllers`, `Effectors`, `Configurators`, `QoS Monitors`, and `Resource Monitors`, can be plugged directly into RACE without modifying RACE's existing architecture.

III.3 Empirical Results and Analysis

This section presents the design and results of experiments that evaluate the scalability of RACE. These experiments validate our claims in Section III.2 that RACE is an scalable adaptive resource management framework.

III.3.1 Hardware and Software Testbed

Our experiments were performed on the ISISLab testbed [1] located at Vanderbilt University. The hardware configuration consists of six nodes and hardware configuration of each nodes was the following: 2.8 GHz Intel Xeon dual processor, 1 GB physical memory, 1GHz Ethernet network interface, and 40 GB hard drive. The Redhat Fedora Core release 4 OS with real-time preemption patches [56] was used for all the nodes.

[1]`http://www.dre.vanderbilt.edu/ISISlab`

Our experiments also used CIAO/DAnCE 0.5.10, which is our open-source QoS enabled component middleware that implements the OMG Lightweight CCM [57] and Deployment and Configuration [61] specifications. RACE and our applications used to measure the scalability of RACE are built upon CIAO/DAnCE.

Table 11 summarizes the number of lines of C++ code of various entities in our middleware, RACE, and our test applications, which were measured using SLOCCount[2].

Entity	Total Lines of Source Code
Test Applications	19,875
RACE	157,253
CIAO/DAnCE	511,378

Table 11: Lines of Source Code for Various System Elements

III.3.2 Evaluation of RACE's Scalability

Sections III.2.0.3 and III.2.0.7 claimed that the hierarchical design of RACE's monitors and effectors enables RACE to scale as the number of applications and nodes in the system grows. We validated this claim by studying the impact of increasing number of nodes and applications on RACE's monitoring delay and actuation delay when RACE's monitors and effectors are configured hierarchically and non-hierarchically. As described in Sections III.2.0.3 and III.2.0.7, *monitoring delay* is defined as the time taken to obtain a snapshot of the entire system in terms of resource utilization and QoS and *actuation delay* is defined as the time taken to execute controller recommended adaptation throughout the system.

To measure the monitoring and actuation delays, we instrumented RACE's `Central Monitor` and `Central Effector`, respectively, with ACE High Resolution Timers [66]. The timer in the `Central Monitor` measured the time duration from when requests were sent to individual `Node Monitors` to the time instant when replies from all `Node`

[2]`http://www.dwheeler.com/sloccount`

`Monitors` were received and the data (resource utilization and application QoS) were assembled to obtain a snapshot of the entire system. Similarly, the timer in the `Central Effector` measured the time duration from when system adaptation decisions were received from the `Controller` to the time instant when acknowledgment indicating successful execution of node level adaption from individual `Effectors` (located on each node) were received.

III.3.2.1 Experiment 1: Constant Number of Application and Varying Number of Nodes

This experiment studied the impact of varying number of nodes in the system domain on RACE's monitoring and actuation delay. We present the results obtained from running the experiment with a constant of five applications, each composed of six components (plasma-sensor/camera-sensor, analysis, filter, analysis, compression, communication, and ground), and a varying number of nodes.

Experiment configuration. We varied the number of nodes in the system from one to six. A total of 30 application components were evenly distributed among the nodes in the system. The experiment was composed of two scenarios: (1) hierarchical and (2) non-hierarchical configuration of RACE's monitors and effectors. Each scenario was comprised of seven runs, and number of nodes in the system during each run was constant[3]. During each run, monitoring delay and actuation delay was collected over 50,000 iterations.

Analysis of results. Figures 36 and 37 compare the impact of increasing the number of nodes in the system on RACE's monitoring and actuation delay, respectively, under the two scenarios. Figures 36 and 37 show that monitoring and actuation delays are significantly lower in the hierarchical configuration of RACE's monitors and effectors compared to the non-hierarchical configuration. Moreover, as the number of nodes in the system increases, the increase in monitoring and actuation delays are significantly (*i.e.*, 18% and

[3]As we varied the number of nodes from one to six each scenario had a total of seven runs.

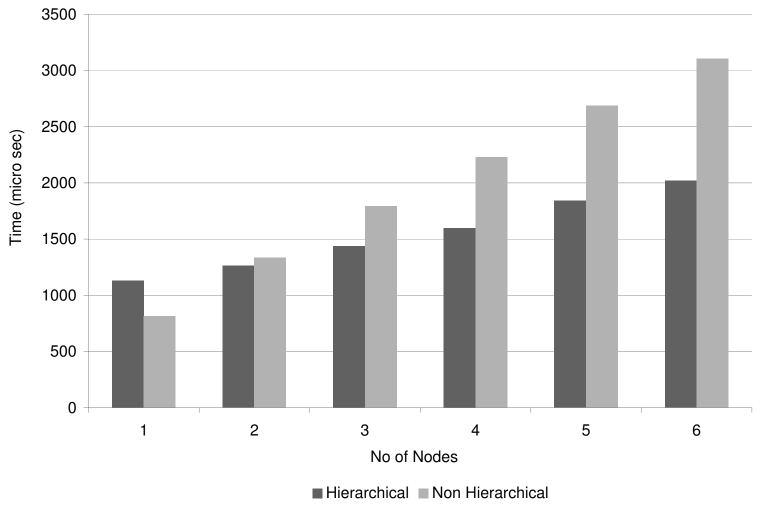


Figure 36: Impact of Increase in Number of Nodes on Monitoring Delay

29%, respectively) lower in the hierarchical configuration compared to the non-hierarchical configuration. This result occurs because individual node monitors and effectors execute in parallel when monitors and effectors are structured hierarchically, thereby significantly reducing monitoring and actuation delay, respectively.

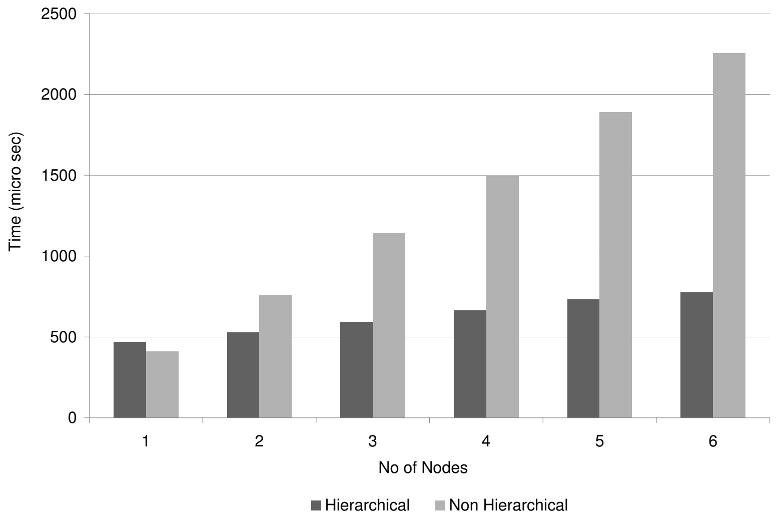


Figure 37: Impact of Increase in Number of Nodes on Actuation Delay

Figures 36 and 37 show the impact on monitoring and actuation delay when the monitors and effectors are structured hierarchically and the number of nodes in the system increase. Although individual monitors and effectors execute in parallel, resource data aggregation and computation of per-node adaptation decisions are centralized by the `Central Monitor` and `Central Effector`, respectively. The results show that this configuration yields a marginal increase in the monitoring and actuation delay (*i.e.*, 6% and 9%, respectively) as the number of nodes in the system increases.

Figures 36 and 37 show that when there is only one node in the system, the performance of the hierarchical configuration of RACE's monitors and effectors is worse than the non-hierarchical configuration. This result measures the overhead associated with the hierarchical configuration. As shown in Figures 36 and 37, however, as the number of nodes in the system increase, the benefit of the hierarchical configuration outweighs this overhead.

III.3.2.2 Experiment 2: Constant Number of Nodes and Varying Number of Applications

This experiment studied the impact of varying the number of applications on RACE's monitoring and actuation delay. We now present the results obtained from running the experiment with six nodes in the system and varying number of applications (from one to five), each composed of six components (plasma-sensor/camera-sensor, analysis, filter, analysis, compression, communication, and ground).

Experiment configuration. We varied the number of applications in the system from one to five. Once again, the application components were evenly distributed among the six nodes in the system. This experiment was composed of two scenarios: (1) hierarchical and (2) non-hierarchical configuration of RACE's monitors and effectors. Each scenario was comprised of five runs, with the number of applications used in each run held constant. As we varied the number of applications from one to five, for each scenario we had a total of

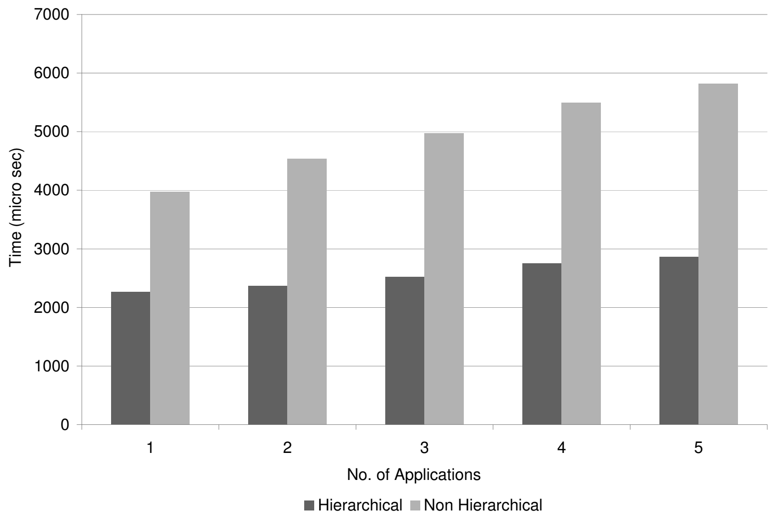


Figure 38: Impact of Increase in Number of Application on Monitoring Delay

five runs. During each run, monitoring delay and actuation delay was collected over 50,000 iterations.

Analysis of results. Figures 38 and 39 compare the impact on increase in number of applications on RACE's monitoring and actuation delay, respectively, under the two scenarios. Figures 38 and 39 show that monitoring and actuation delays are significantly lower under the hierarchical configuration of RACE's monitors and effectors compared with the non-hierarchical configuration. These figures also show that under the hierarchical configuration, there is a marginal increase in the monitoring delay and negligible increase in the actuation delay as the number of applications in the system increase.

These results show that RACE scales well with as the number of nodes and applications in the system increase. The results also show that RACE's scalability is primarily due to the hierarchical design of RACE's monitors and effectors, there by validating our claims in Sections III.2.0.3 and III.2.0.7.

III.3.3 Summary of Experimental Analysis

This section evaluated the performance and scalability of the RACE framework by studying the impact of increase in number of nodes and applications in the system on

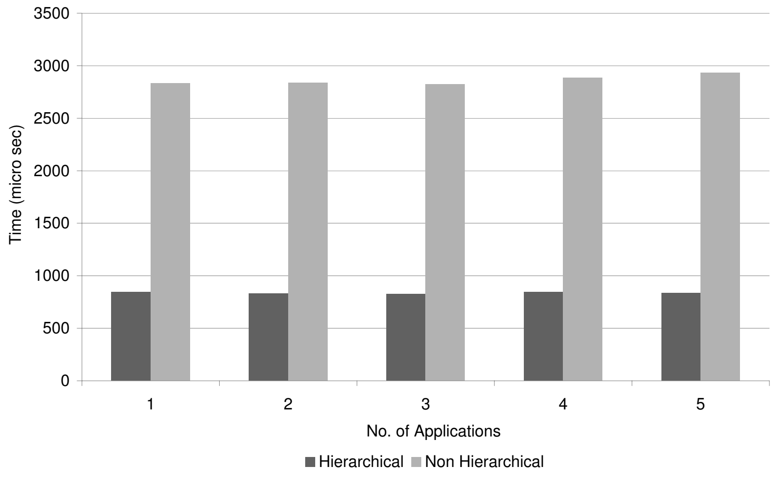


Figure 39: Impact of Increase in Number of Application on Actuation Delay

RACE's monitoring delay and actuation delay. Our results show that RACE is a scalable adaptive resource management framework. From analyzing the results in Sections III.3.2 we observe that RACE scales well as the number of nodes and applications in the system increases. This scalability stems from RACE's the hierarchical design of monitors and effectors, which validates our claims in Sections III.2.0.3 and III.2.0.7.

III.4 Summary

This chapter described the *Resource Allocation and Control Engine* (RACE), which is our adaptive resource management framework that provides end-to-end adaptation and resource management for open DRE systems built atop QoS-enabled component middleware. Open DRE systems built using RACE benefit from the advantages of component-based middleware, as well as QoS assurances provided by adaptive resource management algorithms. We demonstrated how RACE helped resolve key resource and QoS management challenges of open DRE systems. As the elements of the RACE framework are CCM components, RACE itself can be configured using model-driven tools, such as PICML [8]. Moreover, new `InputAdapters`, `Allocators`, `Configurators`, and

`Controllers` can be plugged into RACE using PICML without modifying its architecture. RACE can also be used to deploy, allocate resources to, and manage performance of, applications that are composed at design-time and runtime.

CHAPTER IV

CASE STUDY: MAGNETOSPHERIC MULTI-SCALE MISSION DRE SYSTEM

This chapter presents an overview of NASA's Magnetospheric Multi-scale (MMS) mission [23] as a case study. We describe the resource and QoS management challenges involved in developing the MMS mission using QoS-enabled component middleware, how we applied RACE to addresses these challenges, and we present an empirical evaluation of the performance of the system when it was operated with RACE.

IV.1 MMS Mission System Overview

NASA's MMS mission system is a representative open DRE system consisting of several interacting subsystems (both in-flight and stationary) with a variety of complex QoS requirements. As shown in Figure 40, the MMS mission consists of a constellation of five spacecrafts that maintain a specific formation while orbiting over a region of scientific interest. This constellation collects science data pertaining to the earth's plasma and magnetic activities while in orbit and send it to a ground station for further processing. In the MMS mission spacecrafts, availability of resource such as processing power (CPU), storage, network bandwidth, and power (battery) are limited and subjected to runtime variations. Moreover, resource utilization by, and input workload of, applications that execute in this system can not be accurately characterized *a priori*. These properties make the MMS mission system an open DRE system.

Applications executing in this system can be classified as guidance, navigation, and control (GNC) applications and science applications. The GNC applications are responsible for maintaining the spacecraft within the specified orbit. The science applications are responsible for collecting science-data, compressing and storing the data, and transmitting the stored data to the ground station for further processing.

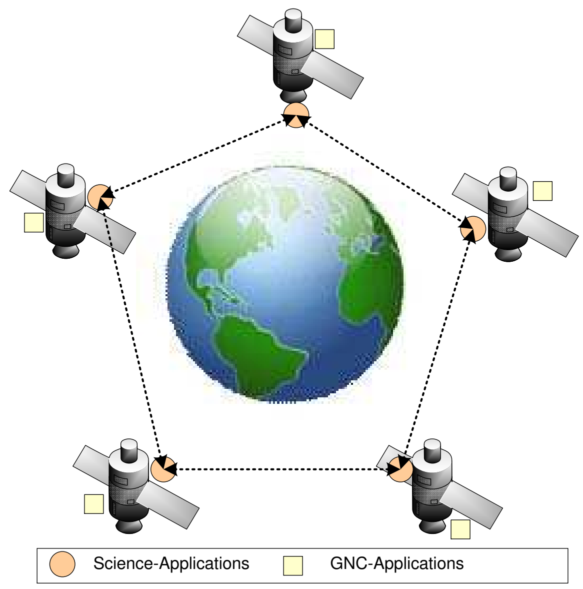


Figure 40: MMS Mission System

As shown in Figure 40, GNC applications are localized to a single spacecraft. Science applications tend to span the entire spacecraft constellation, *i.e.*, all spacecrafts in the constellation have to coordinate with each other to achieve the goals of the science mission. GNC applications are considered *hard real-time* applications (*i.e.*, the penalty of not meeting QoS requirement(s) of these applications is very high, often fatal to the mission), where as science applications are considered *soft real-time* applications (*i.e.*, the penalty of not meeting QoS requirement(s) of these applications is high, but not fatal to the mission).

Science applications operate in three modes: *slow survey*, *fast survey*, and *burst* mode. Science applications switch from one mode to another in reaction to one or more *events of interest*. For example, for a science application that monitors the earth's plasma activity, the *slow* survey mode is entered outside the regions of scientific interests and enables only a minimal set of data acquisition (primarily for health monitoring). The *fast* survey mode is entered when the spacecrafts are within one or more regions of interest, which enables data acquisition for all payload sensors at a moderate rate. If plasma activity is detected while in fast survey mode, the application enters *burst* mode, which results in data collection at the highest data rates. Resource utilization by, and importance of, a science application is determined by its mode of operation, which is summarized by Table 12.

Mode	Relative Importance	Resource Consumption
Slow survey	Low	Low
Fast survey	Medium	Medium
Burst	High	High

Table 12: Characteristics of Science Application

Each spacecraft consists of an on-board intelligent mission planner, such as the *spreading activation partial order planner* (SA-POP) [39] that decomposes overall mission goal(s) into GNC and science applications that can be executed concurrently. SA-POP employs decision-theoretic methods and other AI schemes (such as hierarchical task decomposition) to decompose mission goals into navigation, control, data gathering, and data processing applications. In addition to initial generation of GNC and science applications, SA-POP incrementally generates new applications in response to changing mission goals and/or degraded performance reported by on-board mission monitors.

We have developed a prototype implementation of the MMS mission systems in conjunction with our colleagues at Lockheed Martin Advanced Advanced Technology Center, Palo Alto, California. In our prototype implementation, we used the *Component-Integrated ACE ORB* (CIAO) [81] and *Deployment and Configuration Engine* (DAnCE) [27] as the QoS-enabled component middleware platform. Each spacecraft uses SA-POP as its on-board intelligent mission planner.

IV.2 Adaptive Resource Management Requirements of the MMS Mission System

As discussed in Section III.1.2, the use of QoS-enabled component middleware to develop open DRE systems, such as the NASA MMS mission, can significantly improve the design, development, evolution, and maintenance of these systems. However, when such systems are built in the absence of a adaptive resource frameworks, several key requirements remain unresolved. To motivate the need for RACE, this section presents the key resource and QoS management requirements that we addressed while building our prototype of the MMS mission DRE system.

IV.2.1 Requirement 1: Resource Allocation To Applications

Applications generated by SA-POP are *resource sensitive*, *i.e.*, QoS is affected significantly if an application does not receive the required CPU time and network bandwidth within bounded delay. Moreover, in open DRE systems like the MMS mission, input workload affects utilization of system resources and QoS of applications. Utilization of system resources and QoS of applications may therefore vary significantly from their estimated values. Due to the operating conditions for open DRE systems, system resource availability, such as available network bandwidth, may also be time variant.

A resource management framework therefore needs to (1) monitor the current utilization of system resources, (2) allocate resources in a timely fashion to applications such that their resource requirements are met using resource allocation algorithms such as PBFD [24], and (3) support multiple resource allocation strategies since CPU and memory utilization overhead might be associated with implementations of resource allocation algorithms themselves and select the appropriate one(s) depending on properties of the application and the overheads associated with various implementations. Section IV.3.1 describes how RACE performs on-line resource allocation to application components to addresses this requirement.

IV.2.2 Requirement 2: Configuring Platform-specific QoS Parameters

The QoS experienced by applications depend on various platform-specific real-time QoS configurations including (1) *QoS configuration of the QoS-enabled component middleware*, such as priority model, threading model, and request processing policy, (2) *operating system QoS configuration*, such as real-time priorities of the process(es) and thread(s) that host and execute within the components respectively, and (3) *networks QoS configurations*, such as `diffserv` code-points of the component interconnections. Since these configurations are platform-specific, it is tedious and error-prone for system developers or SA-POP to specify them in isolation.

An adaptive resource management framework therefore needs to provide abstractions that shield developers and/or SA-POP from low-level platform-specific details and define higher-level QoS specification models. System developers and/or intelligent mission planners should be able to specify QoS characteristics of the application such as QoS requirements and relative importance, and the adaptive resource management framework should then configure the platform-specific parameters accordingly. Section IV.3.2 describes how RACE provides higher a level abstractions and shield system developers and SA-POP from low-level platform-specific details to addresses this requirement.

IV.2.3 Requirement 3: Enabling Dynamic System Adaptation and Ensuring QoS Requirements are Met

When applications are deployed and initialized, resources are allocated to application components based on the *estimated* resource utilization and estimated/current availability of system resources. In open DRE systems, however, *actual* resource utilization of applications might be significantly different than their estimated values, as well as availability of system resources vary dynamically. Moreover, for applications executing in these systems, the relation between input workload, resource utilization, and QoS cannot be characterized *a priori*.

An adaptive resource management framework therefore needs to provide monitors that track system resource utilization, as well as QoS of applications, at run-time. Although some QoS properties (such as accuracy, precision, and fidelity of the produced output) are application-specific, certain QoS (such as *end-to-end latency* and throughput) can be tracked by the framework transparently to the application. However, customization and configuration of the framework with domain specific monitors (both platform specific resource monitors and application specific QoS monitors) should be possible. In addition, the framework needs to enable the system to *adapt* to dynamic changes, such as variations in operational conditions, input workload, and/or resource availability. Section IV.3.3 demonstrates how RACE performs system adaptation and ensures QoS requirements of applications are met to address this requirement.

IV.3 Addressing MMS Mission Requirements Using RACE

We now describe how RACE was applied to our MMS mission case study from Section IV.1 and show how it addressed key resource allocation, QoS configuration, and adaptive resource management requirements that we identified in Section IV.1.

IV.3.1 Addressing Requirement 1: Resource Allocation to Applications

RACE's `InputAdapter` parses the metadata that describes the application to obtain the resource requirement(s) of components that make up the application and populates the `E-2-E` IDL structure. The `Central Monitor` obtains system resource utilization/availability information for RACE's `Resource Monitors`, and using this information along with the *estimated* resource requirement of application components captured in the `E-2-E` structure, the `Allocators` map components onto nodes in the system domain based on runtime resource availability.

RACE's `InputAdapter`, `Central Monitor`, and `Allocators` coordinate with one another to allocate resources to applications executing in open DRE systems, thereby addressing the resource allocation requirement for open DRE systems identified in Section IV.2.1.

IV.3.2 Addressing Requirement 2: Configuring Platform-specific QoS Parameters

RACE shields application developers and SA-POP from low-level platform-specific details and defines a higher-level QoS specification model. System developers and SA-POP specify only QoS characteristics of the application, such as QoS requirements and relative importance, and RACE's `Configurators` automatically configures platform-specific parameters appropriately.

For example, consider two science applications – one executing in fast survey mode and one executing in slow survey mode. For these applications, middleware parameters configured by the `Middleware Configurator` includes: (1) CORBA end-to-end priority, which is configured based on execution mode (fast/slow survey) and application period/deadline, (2) CORBA priority propagation model (CLIENT_PROPAGATED / SERVER_DECLARED), which is configured based on the application structure and interconnection, and (3) threading model (single threaded / thread-pool / thread-pool with lanes), which is configured based on number of concurrent peer-components connected to a component. The `Middleware Configurator` derives configuration for such low level platform-specific parameters from application end-to-end structure and QoS requirements.

RACE's `Configurators` provides higher level abstractions and shield system developers and SA-POP from low-level platform-specific details, thus addressing the requirements associated with configuring platform-specific QoS parameters identified in Section IV.2.2.

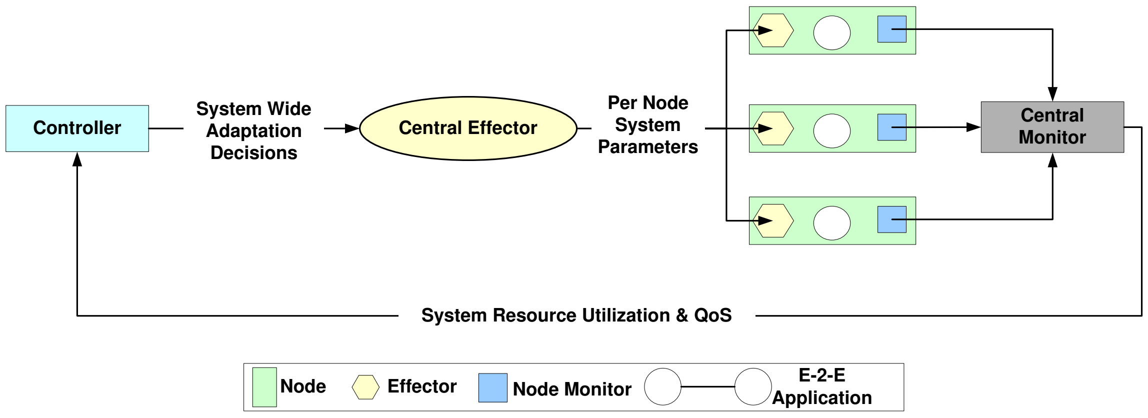

Controller
System Wide Adaptation Decisions
Central Effector
Per Node System Parameters
Central Monitor
System Resource Utilization & QoS
Node
Effector
Node Monitor
E-2-E Application

Figure 41: RACE's Feedback Control Loop

IV.3.3 Addressing Requirement 3: Monitoring End-to-end QoS and Ensuring QoS Requirements are Met

When resources are allocated to components at design-time by system designers using PICML, *i.e.* mapping of application components to nodes in the domain are specified, these operations are performed based on estimated resource utilization of applications and estimated availability of system resources. Allocation algorithms supported by RACE's `Allocators` allocate resources to components based on current system resource utilization and component's estimated resource requirements. In open DRE systems, however, there is often no accurate *a priori* knowledge of input workload, the relationship between input workload and resource requirements of an application, and system resource availability.

To address this requirement, RACE's control architecture employs a feedback loop to manage system resource and application QoS and ensures (1) QoS requirements of applications are met at all times and (2) system stability by maintaining utilization of system resources below their specified utilization set-points. RACE's control architecture features a feedback loop that consists of three main components: `Monitors`, `Controllers`, and `Effectors`, as shown in Figure 41.

`Monitors` are associated with system resources and QoS of the applications and periodically update the `Controller` with the current resource utilization and QoS of applications currently running in the system. The `Controller` implements a particular control algorithm such as EUCON [52], DEUCON [83], HySUCON [41], and FMUF [18], and computes the adaptations decisions for each (or a set of) application(s) to achieve the desired system resource utilization and QoS. `Effectors` modify system parameters, which include resource allocation to components, execution rates of applications, and OS/-middleware/network QoS setting of components, to achieve the controller recommended adaptation.

As shown in Figure 41, RACE's monitoring framework, `Controllers`, and `Effectors` coordinate with one another and the aforementioned entities of RACE to ensure (1) QoS requirements of applications are met and (2) utilization of system resources are maintained within the specified utilization set-point set-point(s), thereby addressing the requirements associated with runtime end-to-end QoS management identified in Section IV.2.3. We empirically validate this in Section IV.4.

IV.4 Empirical Results and Analysis

This section presents the design and results of experiments that evaluate the adaptive resource management capabilities of RACE in the context of our MMS system. This section also validates our claims in Section IV.3 that RACE performs effective end-to-end adaptation and yield a predictable and scalable DRE system under varying operating conditions and input workload.

IV.4.1 Hardware and Software Testbed

Our experiments were performed on the ISISLab testbed[1] at Vanderbilt University. The hardware configuration consists of six nodes, five of which acted as spacecrafts and one that

[1]`www.dre.vanderbilt.edu/ISISlab`

acted as a ground station. The hardware configuration of all the nodes was a 2.8 GHz Intel Xeon dual processor, 1 GB physical memory, 1GHz Ethernet network interface, and 40 GB hard drive. The Redhat Fedora Core release 4 OS with real-time preemption patches [56] was used for all the nodes.

Our experiments also used CIAO/DAnCE 0.5.10, which is our open-source QoS enabled component middleware that implements the OMG Lightweight CCM [57] and Deployment and Configuration [61] specifications. RACE and our DRE system case study are built upon CIAO/DAnCE.

IV.4.2 MMS DRE System Implementation

Science applications executing atop our MMS DRE system are composed of the following components:

- **Plasma sensor component**, which manages and controls the plasma sensor on the spacecraft, collects metrics corresponding to the earth's plasma activity.
- **Camera sensor component**, which manages and controls the high-fidelity camera on the spacecraft and captures images of one or more star constellations.
- **Filter component**, which processes the data from the sensor components to remove any extraneous noise in the collected data/image.
- **Analysis component**, which processes the collected data to determine if the data is of interest or not. If the data is of interest, the data is compressed and transmitted to the ground station.
- **Compression component**, which uses loss-less compression algorithms to compresses the collected data.
- **Communication component**, which transmits the compressed data to the ground station periodically.

- **Ground component**, which received the compressed data from the spacecrafts and stores it for further processing.

Estimated execution times for these components on the system test-bed is shown in Table 13. All these components—except for the ground component—execute on the spacecrafts.[2] Table 14 summarizes the number of lines of C++ code of various entities in our middleware, RACE, and our prototype implementation of the MMS DRE system case study, which were measured using SLOCCount[3].

Component	Estimated Execution Time (msec)
Plasma sensor	35
Camera sensor	40
Ground	50
Filter	55
Analysis	65
Compression	70
Communication	90

Table 13: Estimated Execution Times for Various Application Components

Entity	Total Lines of Source Code
MMS DRE System	19,875
RACE	157,253
CIAO/DAnCE	511,378

Table 14: Lines of Source Code for Various System Elements

IV.4.3 Evaluation of RACE's Adaptive Resource Management Capabilities

We now evaluate the adaptive resource management capabilities of RACE under two scenarios: (1) moderate workload, and (2) heavy workload. Applications executing on our

[2]Our experiments used component emulations that have the same resource utilization characteristics as the original components.

[3]http://www.dwheeler.com/sloccount

prototype MMS mission DRE system were periodic, with deadline equal to their periods. In both the scenarios, we use the deadline miss ratio of applications as the metric to evaluate system performance. For every sampling period of RACE's `Controller`, deadline miss ratio for each application was computed as the ratio of number of times the application's end-to-end latency [4] was greater than its deadline to the number of times the application was invoked.

IV.4.3.1 Summary of Evaluated Scheduling Algorithms

We studied the performance of the prototype MMS system under various configurations: (1) a baseline configuration without RACE and static priority assigned to application components based on Rate Monotonic Scheduling (RMS) [42], (2) a configuration with RACE's Maximum Urgency First (MUF) `Configurator`, and (3) a configuration with RACE's MUF `Configurator` and Flexible MUF (FMUF) [18] `Controller`. The goal of these experiments is not to compare the performance of various adaptive resource management algorithms, such as EUCON [52], DEUCON [83], HySUCON [41], or FMUF. Instead, the goal is to demonstrate how RACE can be used to implement these algorithms and there by meet the system adaptation requirements of open DRE systems.

A disadvantage of RMS scheduling is that it cannot provide performance isolation for higher importance applications [75]. During system overload caused by dynamic increase in the workload, applications of higher importance with a low rate may miss deadlines. Likewise, applications with medium/lower importance but high rates may experience no missed deadlines.

In contrast, MUF provides performance isolation to applications of higher importance by dividing operating system and/or middleware priorities into two classes [75]. All components belonging to applications of higher importance are assigned to the high-priority

[4]The end-to-end latency of an application was obtained from RACE's QoS `Monitors`.

class, while all components belonging to applications of medium/lower importance are assigned to the low-priority class. Components within a same priority class are assigned operating system and/or middleware priorities based on the RMS policy. Relative to RMS, however, MUF may cause priority inversion when an higher importance application has a lower rate than medium/lower importance applications. As a result, MUF may unnecessarily cause an application of medium/lower importance to miss its deadline, even when all tasks are schedulable under RMS.

To address limitations with MUF, RACE's FMUF `Controller` provides performance isolation for applications of higher importance while reducing the deadline misses of applications of medium/lower importance. While both RMS and MUF assign priorities statically at deployment time, the FMUF `Controller` adjusts the priorities of applications of medium/lower importance dynamically based on performance feedback. The FMUF `Controller` can reassign applications of medium/lower importance to the high-priority class when (1) all the applications currently in the high-priority class meet their deadlines while (2) some applications in the low-priority class miss their deadlines. Since the FMUF `Controller` moves applications of medium/lower importance back to the low-priority class when the high-priority class experiences deadline misses it can effectively deal with workload variations caused by application arrivals and changes in application execution times and invocation rates.

IV.4.3.2 Experiment 1: Moderate Workload

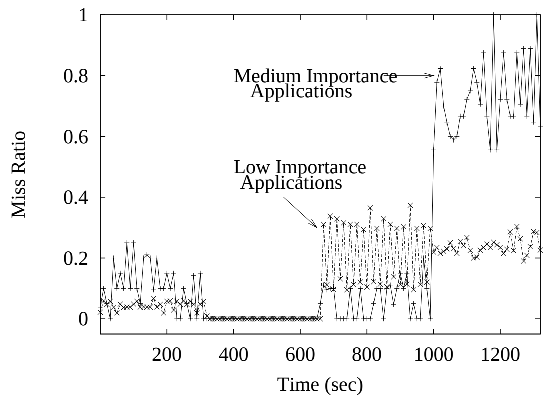
Experiment configuration. The goal of this experiment configuration was to evaluate RACE's system adaptation capabilities under a moderate workload. This scenario therefore employed two of the five emulated spacecrafts, one emulated ground station, and three periodic applications. One application was initialized to execute in fast survey mode and the remaining two were initialized to execute in slow survey mode. As described in Section IV.1, applications executing in fast survey mode have higher relative importance and resource consumption than applications executing in slow survey mode. Each application is subjected to an end-to-end deadline equal to its period. Table 15 summarizes application periods and the mapping of components/applications onto nodes.

Application	Component Allocation			Period (msec)	Mode
	Spacecraft		Ground Station		
	1	2			
1	Communication Plasma-sensor	Analysis Compression	Ground	1000	Fast Survey
2	Analysis Camera-sensor Filter	Communication Compression	Ground	900	Slow Survey
3	Plasma-sensor Camera-sensor	Communication Compression Filter	Ground	500	Slow Survey

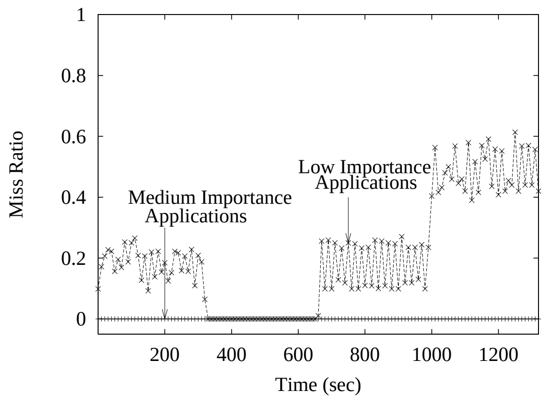
Table 15: Application Configuration under Moderate Workload

The experiment was conducted over 1,400 seconds, and we emulated variation in operating condition, input workload and a mode change by performing the following steps. At time $T = 0sec$, we deployed applications one and two. At time $T = 300sec$, the input workload for all the application were reduced by ten percent, and at time $T = 700sec$ we deployed application three. At $T = 1000sec$, application three switched mode from slow survey to fast survey. To emulate this mode change, we increased the rate (*i.e.* reduced the period) of application three by twenty percent. Since each application was subjected to an end-to-end deadline equal to its period, to evaluate the performance of RACE, we monitored the *deadline miss ratio* of all applications that were deployed.

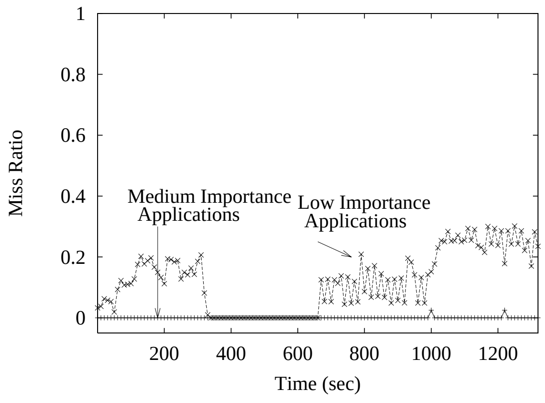
RACE's FMUF `Controller` was used for this experiment since the MMS mission applications described above do not support rate adaptation. RACE is a framework, however, so other adaptation strategies/algorithms, such as HySUCON [41], can be implemented and employed in a similar way. Below, we evaluate the use of FMUF for end-to-end adaptation. Since this paper focuses on RACE—and not the design or evaluation of



(a) Baseline (RMS)



(b) MUF Configurator



(c) MUF Configurator + FMUF Controller

Figure 42: Deadline Miss Ratio Under Moderate Workload

individual control algorithms—we use FMUF as an example to demonstrate RACE's ability to support the integration of feedback control algorithms for end-to-end adaptation in DRE systems. RACE's FMUF controller was configured with the following parameters: sampling period = 10 seconds, $N = 5$, and *threshold = 5%*.

Analysis of results. We now present the results obtained from running the experiment described above on our ISISlab DRE system testbed described in Section IV.4.1. We use deadline miss ratio as the metric to evaluate system performance under varying input workloads and operating conditions.

Figures 42a, 42b, and 42c show the deadline miss ratio of applications when the system was operated under baseline configuration, with RACE's MUF `Configurator`, and with

RACE's MUF `Configurator` along with FMUF `Controller`, respectively. These figures show that under all the three configurations, deadline miss ratio of applications (1) reduced at $T = 300sec$ due to the decrease in the input work load, (2) increased at $T = 700sec$ due to the introduction of new application, and (3) further increased at $T = 1,000sec$ due to the mode change from slow survey mode to fast survey mode. These results demonstrates the impact of fluctuation in input workload and operating conditions on system performance.

Figure 42a shows that when the system was operated under the baseline configuration, deadline miss ratio of medium importance applications (applications executing in fast survey mode) were higher than that of low importance applications (applications executing in slow survey mode) due to reasons explained in Section IV.4.3.1. Figures 42b and 42c show that when RACE's MUF `Configurator` is used (both individually and along with FMUF `Controller`), deadline miss ratio of medium importance applications were nearly zero throughout the course of the experiment. Figures 42a and 42b demonstrate that RACE improves QoS of our DRE system significantly by configuring platform-specific parameters appropriately.

As described in [18], the FMUF `Controller` responds to variations in input workload and operating conditions (indicated by deadline misses) by dynamically adjusting the priorities of the low importance applications (*i.e.*, moving low importance applications into or out of the high-priority class). Figures 42a and 42c demonstrate the impact of the RACE's `Controller` on system performance.

IV.4.3.3 Experiment 2: Heavy Workload

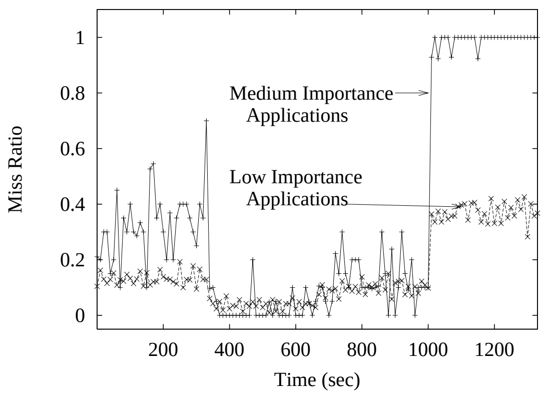
Experiment configuration. The goal of this experiment configuration was to evaluate RACE's system adaptation capabilities under a heavy workload. This scenario therefore employed all five emulated spacecrafts, one emulated ground station, and ten periodic applications. Four of these applications were initialized to execute in fast survey mode and the remaining six were initialized to execute in slow survey mode. Table 16 summarizes the application periods and the mapping of components/applications onto nodes.

Application	Component Allocation						Period (msec)	Mode
	Spacecraft					Ground Station		
	1	2	3	4	5			
1	Communication		Analysis Plasma-sensor	Filter	Compression	Ground	1000	Fast Survey
2	Camera-sensor Compression	Filter Analysis			Communication	Ground	900	Slow Survey
3	Camera-sensor	Plasma-sensor	Communication Compression	Analysis	Filter	Ground	500	Slow Survey
4		Communication	Filter Analysis	Plasma-sensor	Compression	Ground	800	Slow Survey
5	Communication Filter		Camera-sensor	Analysis	Compression	Ground	1200	Slow Survey
6	Analysis	Filter	Communication	Compression	Plasma-sensor	Ground	700	Slow Survey
7	Plasma-sensor	Plasma-sensor	Communication Compression	Analysis	Filter	Ground	600	Fast Survey
8		Communication Filter	Analysis	Plasma-sensor	Compression	Ground	700	Slow Survey
9	Communication Filter		Camera-sensor Plasma-sensor	Analysis	Compression	Ground	400	Fast Survey
10	Compression Filter		Communication Analysis		Plasma-sensor	Ground	700	Fast Survey

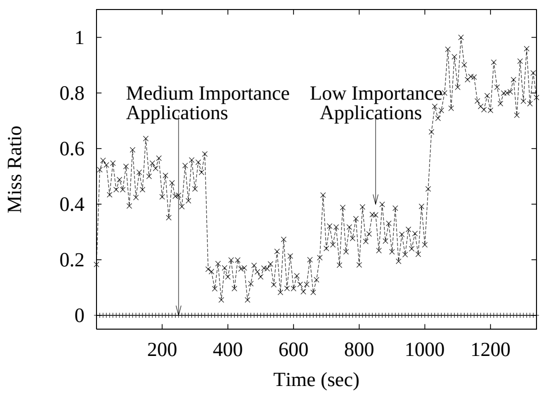
Table 16: Application Configuration under Heavy Workload

The experiment was conducted over 1,400 seconds, and we emulated the variation in operating condition, input workload, and a mode change by performing the following steps. At time $T = 0sec$, we deployed applications one through six. At time $T = 300sec$, the input workload for all the application were reduced by ten percent, and at time $T = 700sec$ we deployed applications seven through ten. At $T = 1,000sec$, applications two through five switched modes from slow survey to fast survey. To emulate this mode change, we increased the rate of applications two through five by twenty percent. RACE's FMUF controller was configured with the following parameters: sampling period = 10 seconds, $N = 5$, and *threshold* = *5%*.

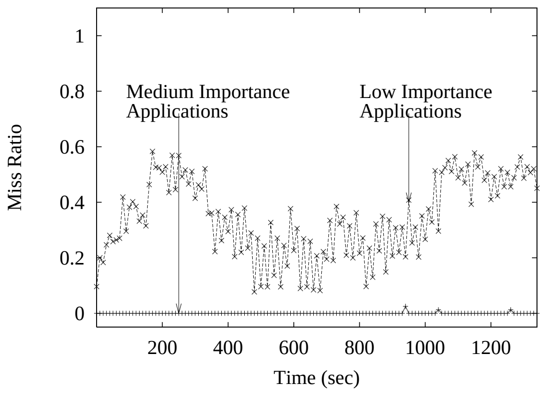
Analysis of results. Figure 43a shows that when the system was operated under the baseline configuration, the deadline miss ratio of the medium importance applications were again higher than that of the low importance applications. Figures 43b and 43c show that when RACE's MUF `Configurator` is used (both individually and along with FMUF `Controller`), deadline miss ratio of medium importance applications were nearly zero throughout the course of the experiment. Figures 43a and 43b demonstrate how RACE



(a) Baseline (RMS)



(b) MUF Configurator



(c) MUF Configurator + FMUF Controller

Figure 43: Deadline Miss Ratio under Heavy Workload

improves the QoS of our DRE system significantly by configuring platform-specific parameters appropriately. Figures 42a and 42c demonstrate that RACE improves system performance (deadline miss ratio) even under heavy workload.

These results show that RACE improves system performance by performing adaptive management of system resources there by validating our claim in Section IV.3.3.

IV.4.4 Summary of Experimental Analysis

This section evaluated the performance and scalability of the RACE framework by studying the impact of increase in number of nodes and applications in the system on RACE's monitoring delay and actuation delay. We also studied the performance of our prototype MMS DRE system with and without RACE under varying operating condition and input workload. Our results show that RACE is a scalable adaptive resource management framework and performs effective end-to-end adaptation and yields a predictable and high-performance DRE system.

From analyzing the results presented in Section IV.4.3, we observe that RACE significantly improves the performance of our prototype MMS DRE system even under varying input workload and operating conditions, thereby meeting the requirements of building component-based DRE systems identified in Section IV.2. These benefits result from configuring platform-specific QoS parameters appropriately and performing effective end-to-end adaptation, which were performed by RACE's `Configurators` and `Controllers`, respectively.

CHAPTER V

CASE STUDY: CONFIGURABLE SPACE MISSION SYSTEMS

In this chapter, we first presents an overview of configurable space mission (CSM) systems, such as the proposed Fractionated Space Mission [14], and uses CSMs as a case study to showcase the challenges of open DRE systems. We then describe how we applied RACE to addresses these challenges. We conclude this chapter by presenting an empirical evaluation of the performance of the system when it was operated with RACE.

V.1 CSM System Overview

A CSM system consists of several interacting subsystems (both in-flight and stationary) executing in an open environment. Such systems consist of a spacecraft constellation that maintains a specific formation while orbiting in/over a region of scientific interest. In contrast to conventional space missions that involve a monolithic satellite, CSMs distribute the functional and computational capabilities of a conventional monolithic spacecraft across multiple modules, which interact via high-bandwidth, low-latency, wireless links.

A CSM system must operate with a high degree of autonomy, adapting to (1) dynamic addition and modifications of user-specified mission goals/objectives; (2) fluctuations in input workload, application resource utilization, and resource availability due to variations in environmental conditions; and (3) complete or partial loss of resources such as computational power and wireless network bandwidth. Moreover, the input workload of—and resource utilization by—applications executing in a CSM system cannot be accurately characterized *a priori*.

The two primary sets of applications executing in an CSM system can be classified as guidance, navigation, and control (GNC) applications and science applications. GNC applications are responsible for maintaining the spacecraft within the specified formation.

Science applications are responsible for collecting science data, processing and analyzing data, storing or discarding the data, and transmitting the stored data to ground stations for further processing. These applications tend to span the entire spacecraft constellation because the fractionated nature of the spacecraft requires a high degree of coordination to achieve mission goals.

GNC applications have *hard* real-time requirements that manage mission-critical attributes of the spacecraft. These applications therefore execute on dedicated resources and cannot countenance any significant adaptation at runtime. In contrast, science applications are generally *soft* real-time applications that can execute on shared resources and can often benefit from runtime adaptation such as fine-tuning of system and/or application properties and parameters.

QoS requirements of science applications can occasionally be unsatisfied without compromising mission success. Moreover, science applications in a CSM system are often periodic, allowing the dynamic modification of their execution rates at runtime. Resource consumption by—and QoS of—these science applications are directly proportional to their execution rates, *i.e.*, a science application executing at a higher rate contributes a higher value to the overall system QoS, but also consumes resources at a higher rate.

V.2 Challenges Associated with the Autonomous Operation of a CSM System

Developing and validating autonomous, open DRE systems, such as CSM systems, presents numerous challenges. This section provides an overview of key adaptation challenges in these systems.

V.2.1 Challenge 1: Dynamic Addition and Modifications of Mission Goals

An operational CSM system can be initialized with a set of goals related to the primary, on-going science objectives. These goals affect the configuration of applications deployed on the system resources, *e.g.*, computational power, memory, and network bandwidth. During normal operation, science objectives could change dynamically and mission goals would be dynamically added and/or modified. In response to dynamic additions/-modifications of science goals, a CSM system must (re)plan its operation to assemble/-modify one or more end-to-end applications (*i.e.*, a set of interacting, appropriately configured application components) to achieve the specified goal under current environmental conditions and resource availability. After one or more applications have been assembled, they will first be allocated system resources and then deployed/initialized atop system resources. Section V.3.1 describes how we resolved this challenge.

V.2.2 Challenge 2: Adapting to Fluctuations in Input Workload, Application Resource Utilization, and Resource Availability

To ensure the stability of open DRE systems, system resource utilization must be kept below specified limits, despite fluctuations in resource availability and demand. Significant under-utilization of system resources is also unacceptable, however, since this can decrease system QoS and increase operational cost. A CSM system must therefore reconfigure application parameters appropriately for these fluctuations (*e.g.*, variations in operational conditions, input workload, and resource availability) to ensure that the utilization of system resources converge to the specified utilization bounds ("set-points"). Autonomous operation of the CSM system therefore requires (1) monitoring of current utilization of system resources, (2) (re)planning for mission goals, considering current environmental conditions and limited resource availability, and (3) timely allocation of system resources to applications that are produced as a result of planning. Section V.3.2 describes how we resolved this challenge.

V.2.3 Challenge 3: Adapting to Complete or Partial Loss of System Resources

In open and uncertain environments, complete or partial loss of system resources—nodes (computational power), network bandwidth, and power—may occur at some point. The autonomous operation of a CSM system requires it to adapt to such failures at run-time, with minimal disruption of the overall mission. Achieving this adaptation requires the ability to optimize overall system expected utility (i.e., the sum of expected utilities of all science applications operating in the system) through prioritizing existing science goals, as well as modifying, removing, and/or redeploying science applications. Consequently, autonomous operation of a CSM system requires (1) monitoring resource liveness, (2) prioritizing mission goals, (3) (re)planning for goals under reduced resource availability, and (4) (re)allocating resources to resulting applications. Section V.3.3 describes how we resolved this challenge.

V.3 Addressing CSM System Challenges

To address the challenges identified in Section V.2, we developed an integrated planning and adaptive resource management architecture, IPAC, which combines an intelligent mission planner [39] and RACE. IPAC, enables self-optimization, self-(re)configuration, and self-organization in open DRE systems by providing decision-theoretic planning, dynamic resource allocation, and runtime system control services. IPAC integrates a planner, resource allocator, a controller, and system monitoring framework, as shown in Figure 44.

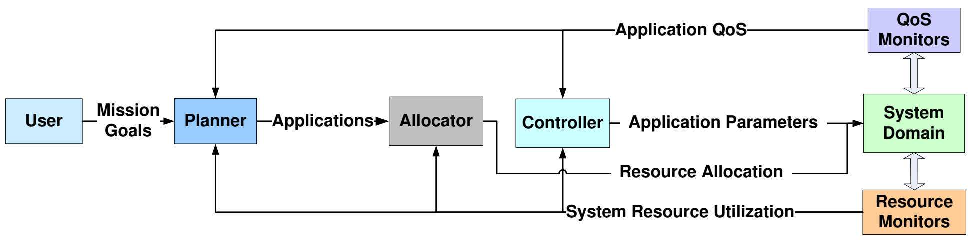


Figure 44: An Integrated Planning, Resource Allocation, and Control (IPAC) Framework for Open DRE Systems

As shown in Figure 44, IPAC uses RACE's resource monitors to track system resource utilization and periodically update the planner, allocator, and controller with current resource utilization (*e.g.*, processor/memory utilization and battery power). RACE's QoS monitors tracks system QoS and periodically updates the planner and RACE's controller with QoS values, such as applications' end-to-end latency and throughput. The planner uses its knowledge of the available components' functional characteristics to dynamically assemble applications (*i.e.*, choose and configure appropriate sets of interacting application components) suitable to current conditions and goals/objectives. During this application assembly, the planner also respects resource constraints and optimizes for overall system expected utility.

IPAC uses RACE's allocators to allocate various domain resources (such as CPU, memory, and network bandwidth) to application components by determining the mapping of components onto nodes in the system domain. After applications have been deployed, IPAC uses RACE's controller to periodically monitor and fine-tune application/system parameters/properties, such as execution rate, to achieve efficient use of system resources.

We now describe how the capabilities offered by IPAC address the system management challenges for open DRE systems identified in Section V.2.

V.3.1 Addressing Challenge 1: Dynamic Addition and Modification of Mission Goals

When IPAC's planner receives a mission goal from a user it assembles an application capable of achieving the provided goal, given current local conditions and resource availability. After the planner assembles an appropriate application, RACE's allocator allocates resources to application components and employs the underlying middleware to deploy and configure the application.

After the application is deployed successfully, the planner updates RACE's controller with the application's metadata including application structure, mapping of allocation components to system resources, and minimum and maximum execution rates. The controller uses this information to dynamically modify system/application parameters (such as execution rates of applications) to accommodate the new application in the system and ensure resources are not over-utilized as a result of this addition. Section V.4.4 empirically evaluates the extent to which IPAC's planning, resource allocation, and runtime system adaptation services can improve system performance in when mission goals are dynamically added to the system or modifications to goals deployed earlier are performed.

V.3.2 Addressing Challenge 2: Adapting to Fluctuations in Input Workload and Application Resource Utilization

IPAC tracks system performance and resource utilization via RACE's resource and QoS monitors. RACE's controller and effectors periodically compute system adaptation decisions and modify system parameters, respectively, to handle minor variations in system resource utilization and performance due to fluctuations in resource availability, input workload, and operational conditions. Section V.4.5 empirically validates how RACE's controller enables the DRE system to adapt to fluctuations in input workload and application resource utilization.

V.3.3 Addressing Challenge 3: Adapting to Complete or Partial Loss of System Resources

When RACE's controller and effectors cannot compensate for changes in resource availability, input workload, and operational conditions (*e.g.*, due to drastic changes in system operating conditions like complete loss of a node), re-planning in the planner is triggered. The planner performs iterative plan repair to modify existing applications to achieve mission goals. Although this re-planning may result in lower expected utility of some applications, it allows the system to optimize overall system expected utility, even in cases of significant resource loss. Section V.4.6 empirically evaluates the extent to which IPAC enables open DRE systems to adapt to loss of system resources.

V.4 Performance Results and Analysis

This section describes experiments and analyzes results that empirically evaluate the performance of our prototype of the configurable space mission (CSM) case study described in Section V.1. These experiments evaluate the extent to which IPAC performs effective end-to-end adaptation, thereby enabling the autonomous operation of open DRE systems. To evaluate how individual services, planning and resource management services, offered by IPAC impact the performance of the system, we ran the experiments in several configurations, *e.g.*, (1) using IPAC with the full set of services (decision-theoretic planning service (planner) and dynamic resource management service (RACE)) enabled and (2) with limited sets of IPAC services enabled.

V.4.1 Hardware and Software Testbed

Our experiments were performed on the ISISLab testbed[1] at Vanderbilt University, which is a cluster consisting of 56 IBM blades powered by Emulab software[2]. Each blade node contains two 2.8 GHz Intel Xeon processors, 1 GB physical memory, 1GHz Ethernet network interface, and 40 GB hard drive. The Redhat Fedora Core release 4 OS with real-time preemption patches [56] was used on all nodes.

We used five blade nodes for the experiments, each acting as a spacecraft in our prototype CSM system. Our middleware platform was CIAO 0.5.10, which is an open-source QoS-enabled component middleware that implements the OMG Lightweight CORBA Component Model (CCM) [57] and Deployment and Configuration [61] specifications. IPAC and the test applications implementing in our CSM system prototype were written in C++ using the CIAO APIs.

[1]`http://www.dre.vanderbilt.edu/ISISlab`
[2]`http://www.emulab.net`

V.4.2 Prototype CSM System Implementation

Mission goals of our prototype CSM system included (1) weather monitoring, (2) monitoring earth's plasma activity, (3) tracking a specific star pattern, and (4) high-fidelity imaging of start constellations. The relative importance of these goals are summarized in Table 17.

#	Goal	Importance
1	Weather Monitoring	100
2	Sunspot Activity Monitoring	80
3	Star Tracking	20
4	Hi-fi Terrestrial Imaging	40

Table 17: Utility of Mission Goals

Applications that achieved these goals were periodic (*i.e.*, applications contained a timer component that periodically triggered the collection, filtration, and analysis of science data) and the execution rate of these applications could be modified at runtime. Table 18 summarizes the number of lines of C++ code of various entities in our CIAO middleware, IPAC framework, and prototype implementation of the CSM DRE system case study, which were measured using SLOCCount[3].

Entity	Total Lines of Source Code
CSM DRE system prototype	18,574
IPAC framework	80,253
CIAO middleware	511,378

Table 18: Lines of Source Code for Various System Elements

[3]http://www.dwheeler.com/sloccount

V.4.3 Experiment Design

As described in Section V.1, a CSM system is subjected to (1) dynamic addition of goals and end-to-end applications, (2) fluctuations in application workload, and (3) significant changes in resource availability. To validate our claim that IPAC enables the autonomous operation of open DRE systems, such as the CSM system, by performing effective end-to-end adaptation, we evaluated performance of our prototype CSM system performance when (1) goals were added at runtime, (2) application workloads were varied at runtime, and (3) a significant drop in available resources occurred due to node failure.

To evaluate the improvement in system performance due to IPAC, we initially indented to compare the system behavior (system resource utilization and QoS) with and without IPAC. However, without IPAC, a planner, a resource allocator, and a controller were not available to the system. Therefore, dynamic assembly of applications that satisfy goals, runtime resource allocation to application components, and online system adaptation to variations in operating conditions, input workload, and resource availability were not possible. In other words, without IPAC our CSM system would reduce to a "static-system" that cannot operate autonomously in open environments.

To evaluate the performance IPAC empirically, we structured our experiments as follows:

- **Experiment 1** presented in Section V.4.4 compares the performance of the system that is subjected to dynamic addition of user goals at runtime when the full set of services (*i.e.*, planning, resource allocation, and runtime control) offered by IPAC are employed to manage the system versus when only the planning and resource allocation services are available to the system.

- **Experiment 2** presented in Section V.4.5 compares the performance of the system that is subjected to fluctuations input workload when the full set of services offered by IPAC are employed to manage the system versus when only planning and resource allocation services are available to the system.

- **Experiment 3** presented in Section II.6.7 compares the performance of the system that is subjected to node failures when the full set of services offered by IPAC are employed to manage the system versus when only resource allocation and control services are available to the system.

For all the experiments, IPAC's planner was configured to use overall system expected utility optimization and respect total system CPU usage constraints. Likewise, the allocator was configured to use a suite of bin-packing algorithms with worst-fit-decreasing and best-fit-decreasing heuristics. Finally, the controller was configured to employ the EUCON [52] control algorithm to compute system adaptation decisions.

V.4.4 Experiment 1: Addition of Goals at Runtime

This experiment compares the performance of the system when the full set of services (*i.e.* planning, resource allocation, and runtime control) offered by IPAC are employed to manage the system versus when only the planning and resource allocation services are available to the system. This experiment also adds user goals dynamically at runtime. The objective is to demonstrate the need for—and empirically evaluate the advantages of—a specialized controller in the IPAC architecture. We use the following metrics to compare the performance of the system under the different service configurations:

1. **System downtime,** which is defined as the duration for which applications in the system are unable to execute due to resource reallocation and/or application redeployment.

2. **Average application throughput,** which is defined as the throughput of applications executing in the system averaged over the entire duration of the experiment.

3. **System resource utilization,** which is measure of the processor utilization on each node in the system domain.

We demonstrate that a specialized controller, such as EUCON, enables the system to adapt more efficiently to fluctuations in system configuration, such as addition of applications to the system. In particular, we empirically show how the service provided by a controller is complementary to the services of both the allocator and the planner.

V.4.4.1 Experiment Configuration

During system initialization, time $T = 0$, the first goal (weather monitoring) was provided to the planner by the user, for which the planner assembled five applications (each with between two and five components). Later, at time $T = 200sec$, the second goal (monitoring earth's plasma activity) goal was provided to the planner, which assembled two applications (with three to four components each) to achieve this goal. Next, at time $T = 400sec$, the third goal (start tracking) was provided to the planner, which assembled one application (with two components) to achieve this goal. Finally, at time $T = 600sec$, the fourth goal (hi-fi imaging) was provided to the planner, which assembled an application with four components to achieve this goal. Table 19 summarizes the provided goals—and the applications deployed corresponding to these goals—as a function of time. Table 20

Time (sec)	Goal	Application #
0 - 200	Weather Monitoring	1 - 5
200 - 400	Sunspot Activity Monitoring	6 - 7
400 - 600	Star Tracking	8
600 - 800	Hi-fi Terrestrial Imaging	9

Table 19: Set of Goals and Corresponding Applications as a Function of Time

summarizes the application configuration, *i.e.*, minimum and maximum execution rates, estimated average resource utilization of components that make up each application, and the ratio of estimated resource utilization between the worst case workload and the average case workload.

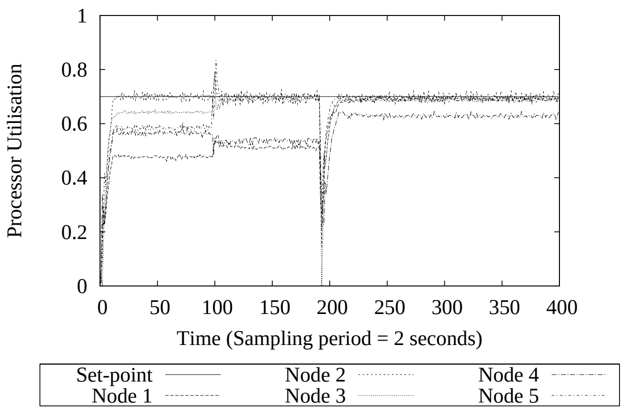
Application	Exec. Rate (Hz)			Net Estimated	Component Average Resource Util.					Util. Ratio
	Min	Max	Init.	Resource Util.	1	2	3	4	5	Average Case : Worst Case
1	15	155	60	0.3	0.15	0.1	0.05	0	0	1 : 1.86
2	35	165	85	0.1	0.05	0.05	0	0	0	1 : 3.00
3	10	140	50	0.5	0.2	0.1	0.1	0.05	0.05	1 : 1.22
4	30	170	80	0.3	0.25	0.05	0	0	0	1 : 3.00
5	35	180	90	0.45	0.2	0.1	0.1	0.05	0	1 : 1.22
6	10	140	65	0.35	0.15	0.1	0.05	0.05	0	1 : 3.00
7	35	170	95	0.35	0.25	0.05	0.05	0	0	1 : 1.86
8	60	95	80	0.35	0.3	0.05	0	0	0	1 : 1.86
9	40	85	60	0.40	0.15	0.10	0.10	0.5	0	1 : 1.20

Table 20: Application Configuration

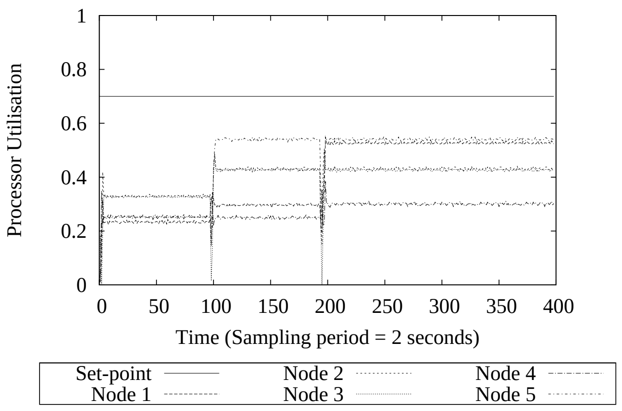
For this experiment, the sampling period of the controller was set to 2 seconds. The processor utilization set-point of the controller, as well as the *bin-size*, of each node was selected to be 0.7, which is slightly lower than RMS [42] utilization bound of 0.77. IPAC allocator was configured to use the standard best-fit-decreasing and worst-fit-decreasing bin-packing heuristics.

V.4.4.2 Analysis of Experiment Results

When IPAC featured the planner, the allocator, and the controller, allocation was performed by the allocator using the average case utilization values due to the availability of the controller to handle workload increases that would result in greater than average resource utilization. When IPAC featured only the planner and the allocator, however, all allocations were computed using the worst case resource utilization values (use of average case utilizations can not be justified because workload increases would overload the system without a controller to perform runtime adaptation). Tables 21 and 22 summarize the initial allocation of components to nodes (for applications 1 - 5 at time $T = 0$ corresponding to the weather monitoring goal), as well as the estimated resource utilization, using average case and worst case utilization values, respectively.



(a) Utilization with the Controller



(b) Utilization without the Controller

Figure 45: Experiment 1: Comparison of Processor Utilization

At time $T = 200sec$, when the applications for the plasma activity monitoring goal were deployed (applications 6 and 7 as specified in Table 20), the system reacted differently when operated with the controller than without it. With the controller, enough available resources were expected (using average case utilization values), so the allocator could incrementally allocate applications 6 and 7 in the system thus required no reallocation or redeployment.

Node	Estimated Utilization	Items (Application, Component)			
1	0.35	(4,1)	(2,1)	(3,5)	
2	0.35	(3,1)	(5,2)	(4,2)	
3	0.35	(5,1)	(5,3)	(5,4)	
4	0.30	(1,1)	(3,3)	(2,2)	
5	0.30	(1,2)	(3,2)	(1,3)	(3,4)

Table 21: Allocation of Applications 1 - 5 using Average Case Utilization

Node	Estimated Utilization	Items (Application, Component)			
1	0.43	(4,1)	(3,5)		
2	0.40	(3,1)	(5,3)	(1,3)	
3	0.39	(5,1)	(5,2)	(3,4)	
4	0.44	(1,1)	(3,3)	(2,2)	(5,4)
5	0.40	(1,2)	(3,2)	(2,1)	(4,2)

Table 22: Allocation of Applications 1 - 5 using Wost Case Utilization

In contrast, when the system operated without the controller, a reallocation was necessary as an incremental addition of applications 6 and 7 to the system was not possible (allocations were based on worst case utilization values). The reallocation of resources requires redeployment of application components and, therefore, increases system/application downtime. Tables 23 and 24 summarize the revised allocation of components to nodes (for applications 1 - 7), as well as the estimated resource utilization, using average case and worst case utilization values, respectively.

Node	Estimated Utilization	Items (Application, Component)				
1	0.45	(4,1)	(2,1)	(3,5)	(6,2)	
2	0.45	(3,1)	(5,2)	(4,2)	(6,3)	(7,2)
3	0.45	(5,1)	(5,3)	(5,4)	(6,4)	(7,3)
4	0.55	(1,1)	(3,3)	(2,2)	(7,1)	
5	0.45	(1,2)	(3,2)	(1,3)	(3,4)	(6,1

Table 23: Allocation of Applications 1 - 7 using Average Case Utilization

At time $T = 400sec$, when the application corresponding to the star tracking goal was provided (application 8), resources were insufficient to incrementally allocate it to the system, both with and without the controller, so reallocation was necessary.

Node	Estimated Utilization	Items (Application, Component)				
1	0.615	(4,1)	(5,3)	(6,4)	(5,4)	
2	0.575	(7,1)	(3,2)	(2,2)	(7,2)	
3	0.605	(6,1)	(1,2)	(3,3)	(4,2)	(7,3)
4	0.610	(3,1)	(1,1)	(2,1)	(1,3)	(3,4)
5	0.610	(5,1)	(6,2)	(5,2)	(6,3)	(3,5)

Table 24: Allocation of Applications 1 - 7 using Wost Case Utilization

When the IPAC was configured without the controller, the allocator was unable to find a feasible allocation using the best-fit decreasing heuristic. However, IPAC's allocator was able to find a feasible allocation using the best-fit decreasing heuristic. Tables 26 and 27 summarize the allocation of components to nodes, as well as the estimated resource utilization, using average case and worst case utilization values, respectively.

At time $T = 600sec$, application corresponding to the hi-fi imaging goal (application 9) had to be deployed. When operating without the controller, it was not possible to find any allocation of all nine applications, and the system continued to operate with only the previous eight applications. In contrast, when the system included the controller, average case utilization values were used during resource allocation, and application 9 was incrementally allocated and deployed in the system.

When the system was operated with the full set of services offered by IPAC the overall system downtime[4] due to resource reallocation and application redeployment was 8534.375 *ms* compared to 15613.162 *ms* when the system was operated without the system adaptation service of IPAC. It is clear that the system downtime is significantly (50%) lower when the system was operating with the full set of services offered by IPAC than when the system was operating without the controller.

From Figure 45, it is clear that system resources are significantly underutilized when

[4]To measure the system downtime, we repeated the experiment over 100 iterations and computed the average system downtime.

operating without the controller but are near the set-point when the controller is used. Underutilization of system resources results in reduced QoS, which is evident from Table 25, showing the overall system QoS.[5]

Application	Average Throughput (Hz)	
	With the Controller	Without the Controller
1	149.973	59.871
2	159.236	84.802
3	100.700	49.624
4	116.453	79.814
5	175.156	89.653
6	25.076	63.212
7	37.370	94.876
8	89.620	79.894
9	40.514	N/A
Entire System	99.344	66.860

Table 25: Experiment 1: Comparison of System QoS

V.4.4.3 Summary

This experiment compared system performance under dynamic addition of mission goals when the full set of IPAC services (*i.e.*, planning, resource allocation, and runtime control) were employed to manage the system versus when only the planning and resource allocation services were available. Significant difference in system evolution were observed due to the fact that when the system was operated without the controller, resources were reallocated more often than when the controller was available. Higher system downtime resulted, further lowering average throughput and resource utilization. Moreover, when the system was operated with the controller, additional mission goals could be achieved by the system, thereby improving the overall system utility and QoS.

From these results, it is clear that without the controller, even dynamic resource allocation is inefficient due to the necessary pessimism in component utilization values (worst case values from profiling). Lack of a controller thus results in (1) under-utilization of

[5]In this system, overall QoS is defined as the total throughput for all active applications.

system resources, (2) low system QoS, and (3) high system downtime. In contrast, when IPAC featured the planner, the allocator, and the controller, resource allocation was significantly more efficient. This efficiency stemmed from the presence of the controller, which ensures system resources are not over-utilized despite workload increases. These results also demonstrate that when IPAC operated with a full set of services it enables the efficient and autonomous operation of the system despite runtime addition of goals.

V.4.5 Experiment 2: Varying Input Workload

This experiment executes an application corresponding to the weather monitoring, monitoring earth's plasma activity, and star tracking goals (applications 1 - 8 described in Table 20), where the input workload is varied at runtime. This experiment demonstrates the adaptive resource management capabilities of IPAC under varying input workload. We compare the performance of the system when the full set of services offered by IPAC (*i.e.*, planning, resource allocation, and runtime control) are employed to manage the system versus when only planning and resource allocation services are available to the system. We use deadline miss ratio, average application throughput and system resource utilization as metrics to empirically compare the performance of the system under each service configuration.

V.4.5.1 Experiment Configuration

At time $T = 0$, the system was initialized with applications 1 - 8 as specified in Table 20. Upon initialization, applications execute at their initialization rate specified in Table 20. When IPAC featured the planner, the allocator, and the controller, allocation was performed by the allocator using the average case utilization values due to the availability of the controller to handle workload increases that would result in greater than average resource utilization. When IPAC featured only the planner and the allocator, however, all allocations were computed using the worst case resource utilization values. Tables 26 and 27 summarize the allocation of components to nodes, as well as the estimated resource utilization, using average case and worst case utilization values, respectively.

Node	Estimated Utilization	Items (Application, Component)				
1	0.55	(8,1)	(3,3)	(2,1)	(3,4)	(6,4)
2	0.55	(4,1)	(1,2)	(5,2)	(3,5)	(7,2)
3	0.55	(7,1)	(3,2)	(5,3)	(4,2)	(7,3)
4	0.55	(3,1)	(1,1)	(6,2)	(5,4)	(8,2)
5	0.50	(5,1)	(6,1)	(1,3)	(2,2)	(6,3)

Table 26: Allocation of Applications 1 - 8 using Average Case Utilization

Node	Estimated Utilization	Items (Application, Component)									
1	0.69	(8,1)	(6,1)	(2,1)							
2	0.70	(4,1)	(7,1)								
3	0.70	(3,1)	(5,1)	(1,1)	(1,3)						
4	0.685	(6,2)	(1,2)	(3,2)	(3,3)	(5,2)	(2,2)				
5	0.695	(5,3)	(4,2)	(6,3)	(6,4)	(7,2)	(7,3)	(8,2)	(3,4)	(3,5)	(5,4)

Table 27: Allocation of Applications 1 - 8 using Wost Case Utilization

Each applicationsŠs end-to-end deadline is defined as $d_i = n_i/r_i(k)$, where n_i is the number of components in application T_i and $r_i(k)$ is the execution rate of application T_i in the k^{th} sampling period. Each end-to-end deadline is evenly divided into sub-deadlines for its components. The resultant sub-deadline of each component equals its period, $1/r(k)$. All application/components meet their deadlines/sub-deadlines if the schedulable utilization bound of RMS [42] is used as the utilization set-point and is enforced on all the nodes.

The sampling period of the controller was set at 2 seconds and the utilization set-point for each node was selected to be 0.7, which is slightly lower than RMS utilization bound. Table 28 summarizes the variation of input workload as a function of time. When the input workload was low, medium, and high, the corresponding resource utilization by application components were their corresponding best case, average case, and worst case values, respectively.

Time (sec)	Input Workload
0 - 150	Low
150 - 450	Medium
450 - 600	High
600 - 900	Medium
900 - 1,000	Low

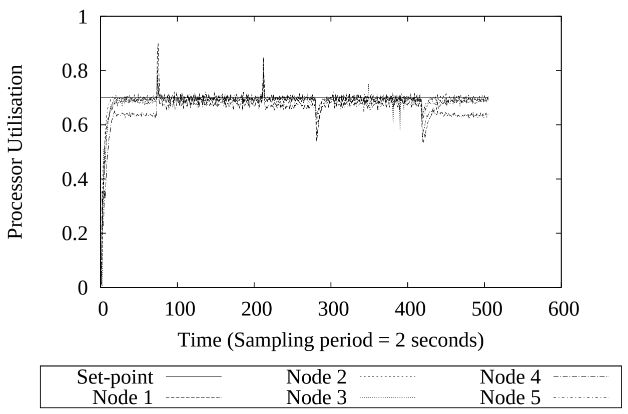
Table 28: Input Workload as a Function of Time

V.4.5.2 Analysis of Experiment Results

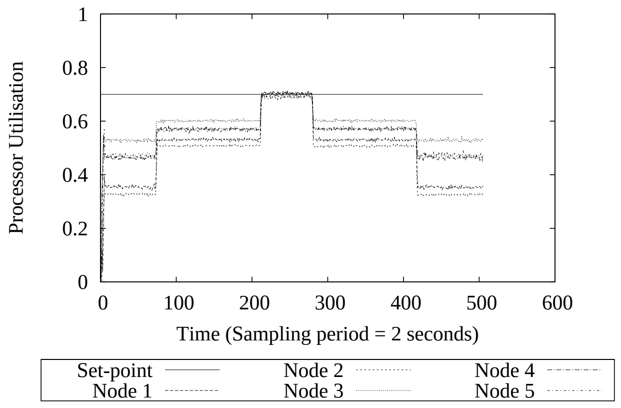
When the IPAC controller is available to the system it dynamically modifies the execution rates of applications within the bounds $[min, max]$ specified in Table 20 to ensure that the resource utilization on each node converges to the specified set-point of 0.7, despite fluctuations in input workload. When IPAC is *not* configured with the controller (*i.e.*, only the planner and the allocator are available), however, applications execute at their initialization rate specified in Table 20.

Figure 46a, Figure 47a, and Table 28 show the execution of the system when it contains the IPAC controller. During $0 \leq T \leq 150$, when the input workload is low, the controller increases the execution rates of applications such that the processor utilization on each node converges to the desired set-point of 0.7. This behavior ensures effective utilization of system resources. When IPAC does not provide the controller service,however, Figures 46b and 47b show that the applications execute at a constant rate (initialization rate) and system resources are severely underutilized.

When input workload is increased from low to medium, at $T = 150s$, the corresponding increase in the processor utilization can be seen in Figure 46. Figures 46a and 47a show that when IPAC included the controller, although the processor utilization increased above the set-point, within a few sampling periods the controller restored the processor utilization to the desired set-point of 0.7 by dynamically reducing the execution rates of applications. Under both service configuration of IPAC, with the controller and without the controller, the deadline miss ratio was 0 throughout the duration of the experiment. Figure 46a shows that the application deadline miss ratio was unaffected by the short duration during which



(a) With the Controller

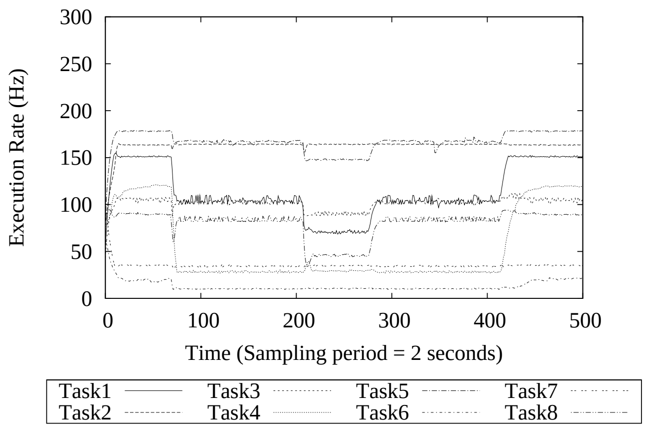


(b) Without the Controller

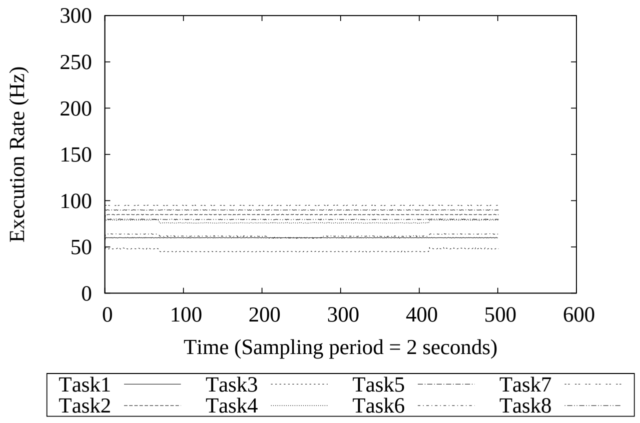
Figure 46: Experiment 2: Comparison of Processor Utilization

processor utilization was above the set-point. Finally, Figure 46b shows that without the controller, the system resources remained under-utilized even after the workload increase.

At $T = 450s$, the input workload was further increased from medium to high. As a result, the processor utilization on all the nodes increased, which is shown in Figure 46. Figures 46a and 47b show that the controller was again able to dynamically modify the application execution rates to ensure that the utilization converged to the desired set-point. Figure 46b shows that when IPAC did not feature the controller, the processor utilization was at the set-point under high workload conditions (corresponding to the worst case resource utilization used to determine the allocation of components to processors in that case).



(a) With the Controller



(b) Without the Controller

Figure 47: Experiment 2: Comparison of Application Execution Rates

At $T = 600s$, when the input workload was reduced from high to medium, from Figure 46 it can be seen that the processor utilization on all the nodes decreased. When IPAC included the controller, however, the controller restored the processor utilization to the desired set-point of 0.7 within a few sampling periods. Without the controller, processor utilization remained significantly lower than the set-point. Similarly, at $T = 900s$, the input workload was further reduced from medium to low, and Figure 46 shows another decrease in processor utilization across all nodes. When IPAC featured the controller, processor utilization again returned to the desired set-point within a few sampling periods. Without the controller, processor utilization remained even further below the set-point.

Figure 46 shows that system resources are significantly underutilized when operating without the controller, but are near the set-point when the controller is used. Underutilization of system resources results in reduced QoS, which is evident from Table 29, showing the overall system QoS.[6] In contrast, when IPAC featured the controller, the application execution rates were dynamically modified to ensure utilization on all the nodes converged to the set-point, resulting in more effective utilization of system resources and higher QoS.

Application	Average Throughput (Hz)	
	With the Controller	Without the Controller
1	113.17	59.930
2	162.817	84.903
3	101.240	45.964
4	54.507	76.909
5	166.959	89.905
6	13.460	62.088
7	35.219	94.896
8	80.019	79.702
Entire System	90.923	74.287

Table 29: Experiment 2: Comparison of System QoS

V.4.5.3 Summary

This experiment compared system performance during input workload fluctuations when the system was operated with the full set of IPAC services (*i.e.* planning, resource allocation, and runtime control) versus when only the planning and resource allocation services were available to the system. The results show how IPAC and its controller (1) ensures system resources are not over-utilized, (2) improves overall system QoS, and (3) enables

[6]In this system, overall QoS is defined as the total throughput for all active applications.

the system to adapt to drifts/fluctuations in utilization of system resources by *fine-tuning* application parameters.

V.4.6 Experiment 3: Varying Resource Availability

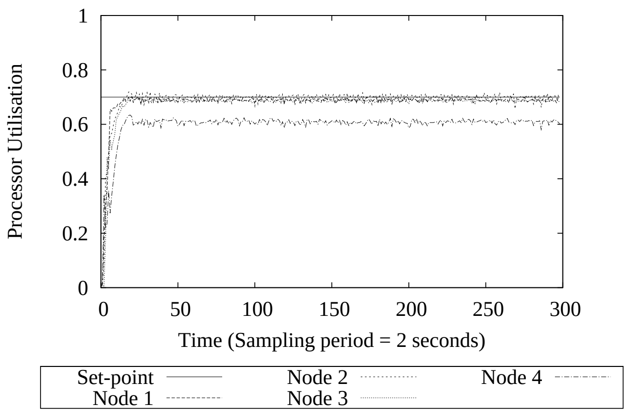
This experiment demonstrate the need for—and advantages of—a planner in our IPAC architecture. It also demonstrates that although a specialized controller can efficiently handle minor fluctuations in the system, it is unable to handle major fluctuations in the system, such as loss of one or more nodes in the system.

We compare the performance of the system when the full set of services offered by IPAC (*i.e.*, planning, resource allocation, and runtime control) are employed to manage the system versus when only resource allocation and control services are available to the system. We use system expected utility and system resource utilization as metrics to empirically compare the performance of the system under each service configuration.

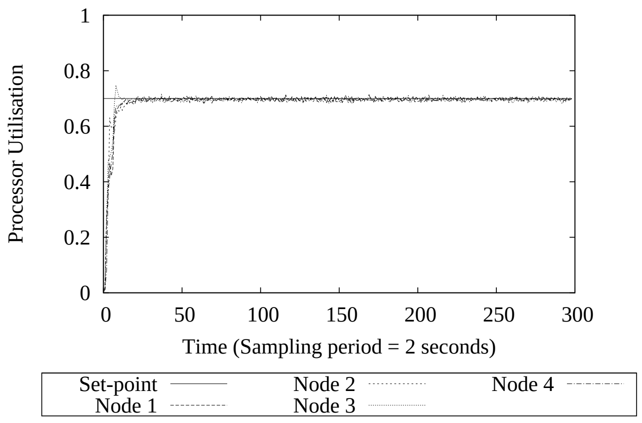
V.4.6.1 Experiment Configuration

For this experiment, the goals provided to the system were (1) weather monitoring, (2) sunspot monitoring, (3) star-tracking, and (4) hi-fi imaging goals. The sampling period of the controller was set to be 2 seconds. The processor utilization set-point of the controller, as well as the *bin-size*, of each node was selected to be 0.7. Under both configurations of IPAC (*i.e.*, (1) when IPAC featured the planner, allocator, and controller and (2) when IPAC featured only the allocator and the controller), allocation was performed by the allocator using the average case utilization values due to the availability of the controller to handle workload increases that would result in greater than average resource utilization.

When IPAC featured only the allocator and the controller, the allocator is augmented such that if it is unable to allocate all applications given the reduced system resources, the allocator incrementally removes applications from consideration by lowest *utility density* until a valid allocation can be found. We define utility density as the expected utility of the application divided by its expected resource usage.



(a) With the Planner



(b) Without the Planner

Figure 48: Experiment 3: Comparison of Processor Utilization

V.4.6.2 Analysis of Experiment Results

When IPAC featured only the allocator and the controller, the complete loss of a node triggered reallocation by the allocator. With the reduced system resource, however, the allocator was able to allocate applications corresponding to the weather monitoring, plasma monitoring, and hi-fi imaging goals only.

In contrast, when IPAC featured the planner, the allocator, and the controller, the complete loss of a node triggers re-planning in the planner. The planner then assembled a new set of applications, taking into account the significant reduction in system resources. Although some applications had a lower expected utility than the original ones, all four goals were still achieved with the resources of the four remaining nodes.

Figure 48 shows that both with and without the planner, the controller ensures that the resource utilization on all the nodes are maintained within the specified bounds.

Table 30 compares the utility of the system when IPAC did/did-not feature the planner. This figure shows how system adaptations performed by the planner in response to failure of a node result in higher system utility compared to the system adaptation performed by just the allocator and the controller. The results Table 30 occur because IPAC's planner was

Application	Expected Utility	
	With Planner	Without Planner
1	18	18
2	6	6
3	30	30
4	20	20
5	26	26
6	38	40
7	36	40
8	16	–
9	40	40
Entire System	230	220

Table 30: Experiment 3: Comparison of System Utility

able to assemble modified applications for some mission goals (corresponding to applications 6, 7, and 8), albeit with somewhat lower expected utility, whereas the allocator had to completely remove an application to meet the reduced resource availability.

V.4.6.3 Summary

This experiment shows that although a specialized controller can efficiently handle minor fluctuations in resource availability, it may be incapable of effective system adaptation in the case of major fluctuations, such as loss of one or more nodes in the system. Even with the addition of an intelligent resource allocation scheme, system performance and utility may suffer unnecessarily during major fluctuations in resource availability. In contrast, IPAC's planner has knowledge of system component *functionality* and desired mission goals. As a result, it can perform more effective system adaptation in the face of major fluctuations, such as the loss of a system node.

CHAPTER VI

CASE STUDY: SEAMONSTER SENSOR-WEB

In this chapter, we first presents an overview of the SEAMONSTER sensor-web system [28]. We use this system as a case study to showcase the resource management challenges of large scale open DRE systems. We then describe how we applied RACE to addresses these challenges. We conclude this chapter by presenting an empirical evaluation of the performance of the system when it was operated with RACE.

VI.1 SEAMONSTER Sensor-web Overview

Sensor-webs [25] are large scale open DRE systems consisting of several interacting subsystems and enable the study of scientific and environmental activities, such as weather monitoring/forecasting, ecosystem monitoring, and monitoring of earth's geological activities, in real-time. Sensor-webs also facilitate the real-time analysis and recovery of large volumes of collected scientific data.

One such sensor-web is the SEAMONSTER sensor-web [28]. Currently, the primary focus of SEAMONSTER sensor-web is to monitor geological activities occurring in the Lemon Creek watershed near Juneau, Alaska. The objective of this sensor-web is to monitor and collect data regarding glacier dynamics and mass balance, watershed hydrology, coastal marine ecology, and human impact/hazards in and around the Lemon Creek watershed. The collected data is used to study the correlation between hydrology, glacier velocity, and temperature variation at the Lemon Creek watershed.

The SEAMONSTER sensor-web is comprised of multiple groups of sensors that are deployed "in the field" and collect data of scientific interest. The data collected by multiple sensor groups are relayed to a cluster of servers via both wired and wireless network for processing, correlation, and analysis. These data processing applications are built atop the

Component-Integrated ACE ORB (CIAO) [81] and *Deployment and Configuration Engine* (DAnCE) [27] QoS-enabled component middleware platform.

Scientific data collected by the sensors are passed to data processing applications that execute at the server cluster. Data processing applications may be added or removed to/-from the server cluster during normal operation. The resource utilization by these applications can not be accurately characterized *a priori* as it depends on the input workload of these applications, which in turn is affected by a plethora of environmental conditions and activities. For example, during nominal operation of the SEAMONSTER sensor web, only a subset of the sensors are operational (primarily for baseline monitoring of the Lemon Creek Glacier and Lemon Creek watershed area). Therefore, the input workload of the applications processing the collected data is minimal. However, when evidence is detected that the glacial lake on Lemon Creek Glacier is draining, most or all of the sensors in the sensor web transition to an operational state and much larger quantities of sensor data are collected to allow in-depth analysis of the effects of the lake draining through the glacier into Lemon Creek. During this event, input workload of the data processing applications are significantly higher than during normal operation.

VI.2 Adaptive Resource Management Requirements of the SEAMONSTER Sensor-web

As discussed in Section III.1.2, the use of QoS-enabled component middleware to develop open DRE systems, such as the SEAMONSTER sensor-web, can significantly improve the design, development, evolution, and maintenance of these systems. However, when such systems are built in the absence of a adaptive resource frameworks, several key requirements remain unresolved. To motivate the need for RACE, this section presents the key resource and QoS management requirements that we addressed while building the SEAMONSTER sensor-web.

VI.2.1 Requirement 1: Online Resource Allocation To Data Processing Applications

Data processing applications executing in the server cluster are *resource sensitive*, *i.e.*, QoS of the sensor-web is affected significantly if an application does not receive the required CPU time and network bandwidth within bounded delay. Moreover, in open DRE systems like the SEAMONSTER sensor-web, input workload affects utilization of system resources and QoS of applications. Utilization of system resources and QoS of applications may therefore vary significantly from their estimated values.

A resource management framework therefore needs to monitor the current utilization of system resources and allocate resources in a timely fashion to applications such that their resource requirements are met using resource allocation algorithms such as PBFD [24]. Section VI.3.1 describes how RACE performs on-line resource allocation to application components to addresses this requirement.

VI.2.2 Requirement 2: Enabling the Sensor-web to Dynamically Adapt to Fluctuations in Input Workload

When applications are deployed and initialized, resources are allocated to application components based on the *estimated* resource utilization and estimated/current availability of system resources. In open DRE systems, however, *actual* resource utilization of applications might be significantly different than their estimated values. Moreover, for applications executing in these systems, the relation between input workload, resource utilization, and QoS cannot be characterized *a priori*.

An adaptive resource management framework therefore needs to provide monitors that track system resource utilization, as well as QoS of applications, at run-time. Although some QoS properties (such as accuracy, precision, and fidelity of the produced output) are application-specific, certain QoS (such as *end-to-end latency* and throughput) can be tracked by the framework transparently to the application. However, customization and configuration of the framework with domain specific monitors (both platform specific resource monitors and application specific QoS monitors) should be possible. In addition, the framework needs to enable the system to *adapt* to dynamic changes, such as variations in operational conditions and/or input workload. Section VI.3.2 demonstrates how RACE performs system adaptation and utilization of system resources are maintained within the specified utilization set-point set-point(s) to address this requirement.

VI.3 Addressing SEAMONSTER Requirements Using RACE

We now describe how RACE was applied to the SEAMONSTER sensor-web described in Section VI.1 and show how it addressed key resource allocation and adaptive resource management requirements that we identified in Section VI.2.

VI.3.1 Addressing Requirement 1: Online Resource Allocation

First, RACE's, using its `InputAdapter`, parses the metadata that describes the application to obtain the resource requirement(s) of components that make up the application. The `Central Monitor` obtains system resource utilization/availability information for RACE's `Resource Monitors`, and using this information along with the *estimated* resource requirement of application components captured in application's metadata, the `Allocators` map components onto nodes in the system domain based on runtime resource availability.

RACE's `InputAdapter`, `Central Monitor`, and `Allocators` coordinate with one another to allocate resources to applications executing in open DRE systems, thereby addressing the resource allocation requirement for open DRE systems identified in Section VI.2.1.

VI.3.2 Addressing Requirement 2: Runtime System Adaptation

Allocation algorithms supported by RACE's `Allocators` allocate resources to components based on current system resource utilization and component's estimated resource requirements. In open DRE systems, however, there is often no accurate *a priori* knowledge of input workload and the relationship between input workload and resource requirements of an application.

To address this requirement, RACE's control architecture employs a feedback loop to manage system resource and application QoS and ensures (1) QoS requirements of applications are met at all times and (2) system stability by maintaining utilization of system resources below their specified utilization set-points. RACE's control architecture features a feedback loop that consists of three main components: `Monitors`, `Controllers`, and `Effectors`.

`Monitors` are associated with system resources and QoS of the applications and periodically update the `Controller` with the current resource utilization and QoS of applications currently running in the system. The `Controller` implements a particular control algorithm such as EUCON [52], DEUCON [83], HySUCON [41], and FMUF [18], and computes the adaptations decisions for each (or a set of) application(s) to achieve the desired system resource utilization and QoS. `Effectors` modify system parameters, which include resource allocation to components, execution rates of applications, and OS/-middleware/network QoS setting of components, to achieve the controller recommended adaptation.

RACE's monitoring framework, `Controllers`, and `Effectors` coordinate with one another and the aforementioned entities of RACE to ensure (1) QoS requirements of applications are met and (2) utilization of system resources are maintained within the specified utilization set-point set-point(s), thereby addressing the requirements associated with runtime end-to-end QoS management identified in Section VI.2.2. We empirically validate this in Section VI.4.

VI.4 Performance Results and Analysis

This section presents the design and results of experiments that evaluate the adaptive resource management capabilities of RACE in the context of the SEAMONSTER sensor-web. This section also validates our claims in Section VI.3 that RACE performs effective end-to-end adaptation and yield a predictable and scalable DRE system under varying operating conditions and input workload.

VI.4.1 Hardware and Software Testbed

Our experiments were performed on the ISISLab testbed[1] at Vanderbilt University, which is a cluster consisting of 56 IBM blades powered by Emulab software[2]. Each blade node contains two 2.8 GHz Intel Xeon processors, 1 GB physical memory, 1GHz Ethernet network interface, and 40 GB hard drive. The Redhat Fedora Core release 4 OS with real-time preemption patches [56] was used on all nodes.

We used five blade nodes for the experiments to emulate the server cluster of our prototype SEAMONSTER sensor-web. Our middleware platform was CIAO 0.5.10, which is an open-source QoS-enabled component middleware that implements the OMG Lightweight CORBA Component Model (CCM) [57] and Deployment and Configuration [61] specifications.

VI.4.2 System Implementation and Experiment Design

Data processing application that executed on our prototype SEAMONSTER sensor-web can be classified as (1) glacier dynamics monitoring, (2) watershed hydrology analysis, and (3) coastal marine ecology analysis applications. These applications were periodic (*i.e.*, applications contained a timer component that periodically triggered the collection, filtration, and analysis of science data) and the execution rate of these applications could

[1]`http://www.dre.vanderbilt.edu/ISISlab`
[2]`http://www.emulab.net`

be modified at runtime. Table 31 summarizes the number of lines of C++ code of various entities in our CIAO middleware, RACE, and our implementation of the data processing applications that executed on the prototype SEAMONSTER sensor-web, which were measured using SLOCCount[3].

Entity	Total Lines of Source Code
Data processing applications	18,574
RACE framework	157,253
CIAO middleware	511,378

Table 31: Lines of Source Code for Various System Elements

As described in Section VI.1, the SEAMONSTER sensor-web is subjected fluctuations in application workload. To validate our claim that RACE enables the autonomous operation of open DRE systems, such as the SEAMONSTER sensor-web, by performing effective end-to-end adaptation, we evaluated performance of our prototype SEAMONSTER sensor-web performance when application workloads were varied at runtime. Our experiment compares the performance of the system that is subjected to fluctuations input workload when the system is operated with and without RACE. As execution rates of applications that executed in this system could be dynamically modified at runtime, RACE was configured to employ the EUCON [52] control algorithm to compute system adaptation decisions.

VI.4.3 Evaluation of RACE's Adaptive Resource Management Capabilities

In this experiment input workload to data processing applications were varied at runtime. This experiment demonstrates the adaptive resource management capabilities of RACE under varying input workload. We compare the performance of the system when it was operated with and without RACE. We use deadline miss ratio, average application

[3]http://www.dwheeler.com/sloccount

throughput and system resource utilization as metrics to empirically compare the performance of the system under each service configuration.

VI.4.3.1 Experiment Configuration

Application	Exec. Rate (Hz)			Net Estimated	Component Average Resource Util.				
	Min	Max	Init.	Resource Util.	1	2	3	4	5
1	15	155	60	0.3	0.15	0.1	0.05	0	0
2	35	165	85	0.1	0.05	0.05	0	0	0
3	10	140	50	0.5	0.2	0.1	0.1	0.05	0.05
4	30	170	80	0.3	0.25	0.05	0	0	0
5	35	180	90	0.45	0.2	0.1	0.1	0.05	0
6	10	140	65	0.35	0.15	0.1	0.05	0.05	0
7	35	170	95	0.35	0.25	0.05	0.05	0	0

Table 32: Application Configuration

At time $T = 0$, the system was initialized the applications specified in Table 32 to perform glacier dynamics monitoring, watershed hydrology analysis, and coastal marine ecology analysis. Upon initialization, applications execute at their initialization rate specified in Table 32. Each applicationsŠ end-to-end deadline is defined as $d_i = n_i/r_i(k)$, where n_i is the number of components in application T_i and $r_i(k)$ is the execution rate of application T_i in the k^{th} sampling period. Each end-to-end deadline is evenly divided into sub-deadlines for its components. The resultant sub-deadline of each component equals its period, $1/r(k)$. All application/components meet their deadlines/sub-deadlines if the schedulable utilization bound of RMS [42] is used as the utilization set-point and is enforced on all the nodes.

The sampling period of the controller was set at 2 seconds and the utilization set-point for each node was selected to be 0.7, which is slightly lower than RMS utilization bound. Table 33 summarizes the variation of input workload as a function of time. When the input workload was low, medium, and high, the corresponding resource utilization by application components were their corresponding best case, average case, and worst case values, respectively.

Sampling Period	Input Workload
0 - 50	Low
50 - 150	Medium
150 - 250	High
250 - 350	Medium
350 - 400	Low

Table 33: Input Workload as a Function of Time

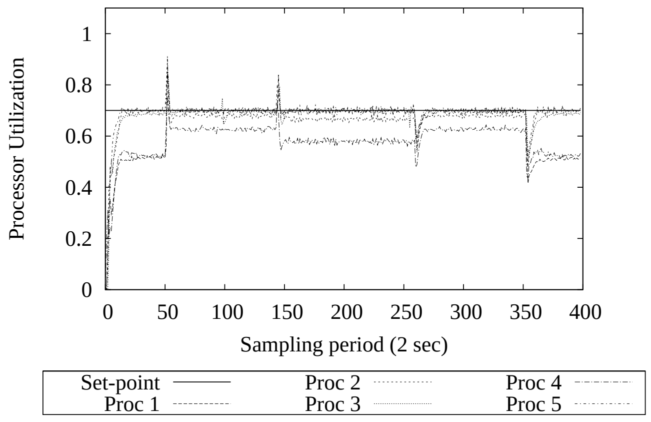
VI.4.3.2 Analysis of Experiment Results

When RACE is available to the system it dynamically modifies the execution rates of applications within the bounds $[min, max]$ specified in Table 32 to ensure that the resource utilization on each node converges to the specified set-point of 0.7, despite fluctuations in input workload. When the system operated without RACE, however, applications execute at their initialization rate specified in Table 32.

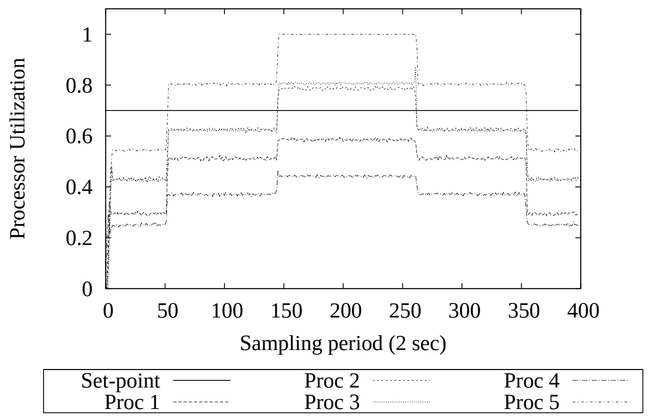
Figure 49a, Figure 50a, and Table 33 show the execution of the system when RACE is employed. During $0 \leq T \leq 100$, when the input workload is low, the controller increases the execution rates of applications such that the processor utilization on each node converges to the desired set-point of 0.7. This behavior ensures effective utilization of system resources. When RACE is not used, however, Figures 49b and 50b show that the applications execute at a constant rate (initialization rate) and system resources are severely underutilized.

When input workload is increased from low to medium, at $T = 100s$, the corresponding increase in the processor utilization can be seen in Figure 49. Figures 49a and 50a show that when RACE is used, although the processor utilization increased above the set-point, within a few sampling periods the controller restored the processor utilization to the desired set-point of 0.7 by dynamically reducing the execution rates of applications. The deadline miss ratio for the entire duration of the experiment was observed to be 0.005 and 0.0184 when the system was operated with and without RACE, respectively. Finally, Figure 49b shows that without RACE, the processor utilization was below the set-point for all the nodes in the system, except for node 5.

At $T = 300s$, the input workload was further increased from medium to high. As a



(a) With RACE

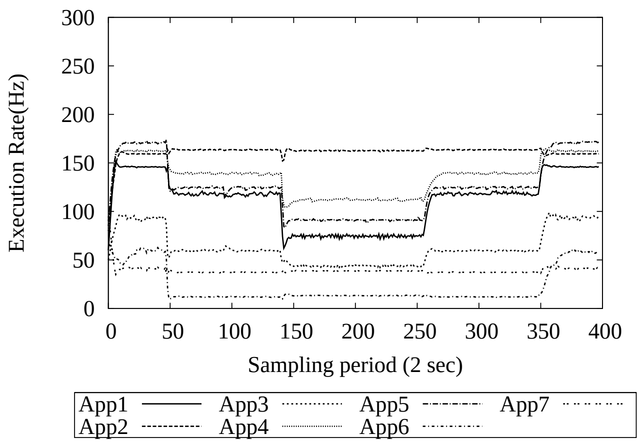


(b) Without RACE

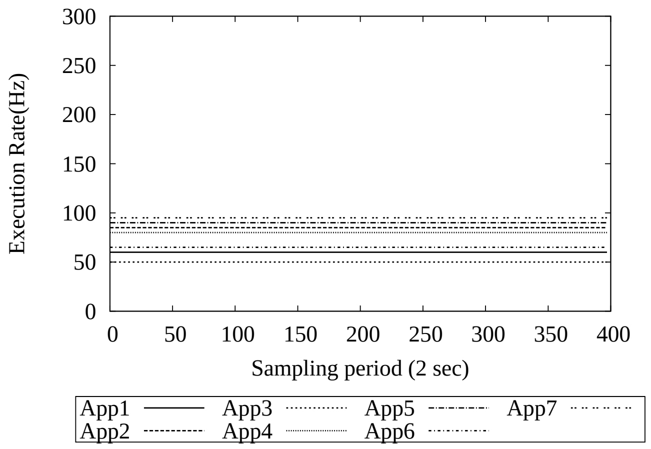
Figure 49: Comparison of Processor Utilizations

result, the processor utilization on all the nodes increased, which is shown in Figure 49. Figures 49a and 50b show that RACE was again able to dynamically modify the application execution rates to ensure that the utilization converged to the desired set-point. Figure 49b shows that without RACE, the processor utilization on most of the nodes in the system was significantly higher than the was at the set-point under high workload conditions.

At $T = 500s$, when the input workload was reduced from high to medium, from Figure 49 it can be seen that the processor utilization on all the nodes decreased. With the system was operated with RACE, however, RACE restored the processor utilization to the



(a) With RACE



(b) Without RACE

Figure 50: Comparison of Application Execution Rates

desired set-point of 0.7 within a few sampling periods. Without RACE, processor utilization for all nodes except node 5 remained significantly lower than the set-point. Similarly, at $T = 700s$, the input workload was further reduced from medium to low, and Figure 49 shows another decrease in processor utilization across all nodes. When the system featured RACE, processor utilization again returned to the desired set-point within a few sampling periods. Without RACE, processor utilization remained even further below the set-point.

Figure 49 shows that system resources are either significantly underutilized or overutilized when operating without RACE, but are near the set-point when RACE is used.

Underutilization and/or over-utilization of system resources results in reduced QoS, which is evident from Table 34, showing the overall system QoS.[4] In contrast, when the system featured RACE, the application execution rates were dynamically modified to ensure utilization on all the nodes converged to the set-point, resulting in more effective utilization of system resources and higher QoS.

Application	Average Throughput (Hz)	
	With RACE	Without RACE
1	110.326	59.930
2	160.891	84.903
3	60.532	45.964
4	133.894	76.909
5	124.232	89.599
6	21.476	63.2362
7	37.264	94.896
Entire System	92.660	74.445

Table 34: Comparison of System QoS

VI.4.3.3 Summary

This experiment compared system performance during input workload fluctuations when the system was operated with and without RACE. The results show how RACE (1) ensures system resources are not over-utilized, (2) improves overall system QoS, and (3) enables the system to adapt to drifts/fluctuations in utilization of system resources by *fine-tuning* application parameters.

[4]In this system, overall QoS is defined as the total throughput for all active applications.

CHAPTER VII

CONCLUDING REMARKS

Open distributed real-time and embedded (DRE) systems require end-to-end QoS enforcement from their underlying operating platforms to operate correctly. These systems often run in environments where resource availability is subject to dynamic change. To meet end-to-end QoS in these dynamic environments, DRE systems can benefit from adaptive resource management architectures that monitors system resources, performs efficient application workload management, and enables efficient resource provisioning for executing applications. Resource management mechanisms based on control-theoretic techniques are emerging as a promising solution to handle the challenges of applications with stringent end-to-end QoS executing in DRE systems. These mechanisms enable adaptive resource management capabilities in open DRE systems and adapt gracefully to fluctuation in resource availability and application resource requirement at runtime.

To address key resource management challenges of open DRE systems, this dissertation presented adaptive resource management algorithms, architectures, and frameworks for large-scale DRE systems. Chapter II described HiDRA, which is a hierarchical distributed resource management architecture based on control-theoretic techniques that provides adaptive resource management, such as resource monitoring and application adaptation, that are key to supporting open DRE systems. Chapter II also presented an evaluation of the performance of HiDRA using a representative target tracking DRE system implemented using RT-CORBA and composed of two types of system resources (computational power at the receiver and wireless network bandwidth) and three applications (UAV data sender/receiver pairs).

Chapter III described RACE, which is an adaptive resource management framework that provides end-to-end adaptation and resource management for open DRE systems built atop QoS-enabled component middleware. Chapter III also demonstrated how RACE helps resolve key resource and QoS management challenges associated with DRE systems. Finally, Chapters IV, V, and VI presented three representative DRE system case studies where we successfully applied RACE. These chapters detailed the adaptive resource management challenges of each DRE system and presented an empirical evaluation of adaptive resource management capabilities of RACE in the context of each DRE system.

VII.1 Lessons Learned

We now summarize the lessons learned from our work on adaptive resource management algorithms, architectures, and frameworks for DRE systems.

VII.1.1 Adaptive Resource Management Algorithms and Architectures

The lessons learned by applying HiDRA to our target tracking system thus far include:

- HiDRA's Control-theoretic approaches yielded in an *adaptive* resource management architecture that can gracefully handle fluctuations in resource availability and/or demand for open DRE systems.

- The formalisms presented in the chapter form the foundation for a resource management framework based on control-theoretic principles that can be used to perform system stability analysis and obtain theoretical assurance about system performance.

- Developing applications in which parameters can be fine-tuned to modify the application operation and utilization of system resources helps achieve higher QoS of applications and enables HiDRA to maintain system resource utilization within desired bounds.

VII.1.2 Adaptive Resource Management Frameworks

The lessons learned in building RACE and applying it three DRE system thus far include:

- **Challenges involved in developing open DRE systems.** Achieving end-to-end QoS in open DRE systems requires adaptive resource management of system resources, as well as integration of a range of real-time capabilities. QoS-enabled middleware, such as CIAO/DAnCE, along with the support of DSMLs and tools, such as PICML, provide an integrated platform for building such systems and are emerging as an operating platform for these systems. Although CIAO/DAnCE and PICML alleviate many challenges in building DRE systems, they do not addresses the adaptive resource management challenges and requirements of open DRE systems. Adaptive resource management solutions are therefore needed to ensure QoS requirements of applications executing atop these systems are met.

- **Decoupling middleware and resource management algorithms.** Implementing adaptive resource management algorithms within the middleware tightly couples the resource management algorithms within particular middleware platforms. This coupling makes it hard to enhance the algorithms without redeveloping significant portions of the middleware. Adaptive resource management frameworks, such as RACE, alleviate the tight coupling between resource management algorithms and middleware platforms and improve flexibility.

- **Design of a framework determines its performance and applicability.** The design of key modules and entities of the resource management framework determines the scalability, and therefore the applicability, of the framework. To apply a framework like RACE to a wide range of open DRE system, it must scale as the number of nodes and application in the system grows. Our empirical studies on the scalability of RACE showed that structuring and designing key modules of RACE (*e.g.*, monitors and effectors) in a hierarchical fashion not only significantly improves the performance of RACE, but also improves its scalability.

- **Need for configuring/customizing the adaptive resource management framework with domain specific monitors.** Utilization of system resources, such as CPU, memory, and network bandwidth, and system performance, such as latency and throughput, can be measured in a generic fashion across various system domains. In open DRE systems, however, the need to measure utilization of domain-specific resources, such as battery utilization, and application-specific QoS metrics, such as the fidelity of the collected plasma data, might occur. Domain-specific customization and configuration of an adaptive resource management framework, such as RACE, should therefore be possible. RACE supports domain-specific customization of its `Monitors`. In future work, we will empirically evaluate the ease of integration of these domain-specific resource entities.

- **Need for selecting an appropriate control algorithm to manage system performance.** The control algorithm that a `Controller` implements relies on certain system parameters that can be fine-tuned/modified at runtime to achieve effective system adaptation. For example, FMUF relies on fine-tuning operating system priorities of processes hosting application components to achieve desired system adaptation; EUCON relies on fine-tuning execution rates of end-to-end applications to achieve the same. The applicability of a control algorithm to a specific domain/scenario is therefore determined by the availability of these runtime configurable system parameters. Moreover, the responsiveness of a control algorithm and the `Controller` in restoring the system performance metrics to their desired values determines the applicability of a `Controller` to a specific domain/scenario. During system design time a `Controller` should be selected that is appropriate for the system domain/scenario.

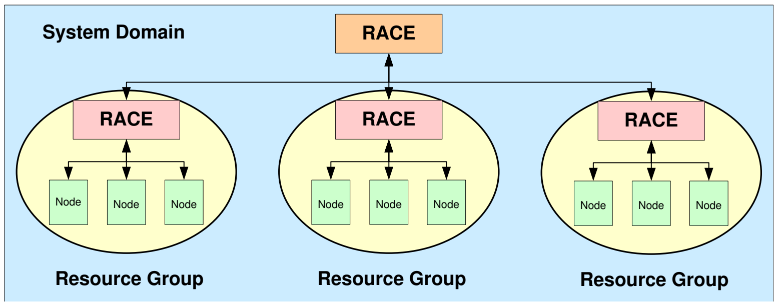


Figure 51: Hierarchical Composition of RACE

- **Need for distributed/decentralized adaptive resource management.** It is easier to design, analyze, and implement *centralized* adaptive resource management algorithms that manage an entire system than it is to design, analyze, and implement *decentralized* adaptive resource management algorithms. As a the size of a system grows, however, centralized algorithms can become bottlenecks since the computation time of these algorithms can scale exponentially as the number of end-to-end applications increases. One way to alleviate these bottlenecks is to partition system resources into *resource groups* and employ hierarchical adaptive resource management, as shown in Figure 51. In our future work we plan to enhance RACE so that a *local* instance of the framework can manage resource allocation, QoS configuration, and runtime adaption within a resource group, whereas a *global* instance can be used to manage the resources and performance of the entire system.

VII.2 Future Research Directions

Based on our experience in designing and developing adaptive resource management algorithms, architectures, and frameworks for DRE systems, we now present some future research directions. Our views and ideas on future research directions are summarized below.

- **Decentralized and/or decoupled resource management algorithms and architectures.** Our solutions to manage resources in DRE systems are built upon the assumption that a centralized *feedback lane* – communication channel between monitors, centralized controller, and effectors – is always open and available. Although this is a reasonable assumption for a significantly large number of DRE systems, this assumption does not hold true for certain flavors of DRE systems where the availability of a communication channel between various pieces of the system is intermittent. Therefore, to broaden the applicability of adaptive resource management solutions, future research is necessary to design and develop adaptive resource management solutions that minimize the reliance on a centralized feedback lane. To address this challenge, one potential approach would involve the design and development of adaptive resource management solutions that (1) are decentralized and/or decoupled and (2) employ multiple individual/local feedback lanes in contrast to existing solutions that a employ centralized controller and rely heavily on the centralized feedback lane.

- **Techniques that enable the coordinated and simultaneous operation of multiple resource management solutions.** Adaptation in open DRE systems can be performed at the various levels. These levels of adaptation include (1) the *system level*, *e.g.*, where applications can be deployed/removed end-to-end to/from the system, (2) the *application structure level*, *e.g.*, where components (or assemblies of components) associated with one or more applications executing in the system can be added, modified, and/or removed, (3) the *resource level*, *e.g.*, where resources can be applied to application components to ensure their timely completion, and (4) the *application parameter level*, *e.g.*, where configurable parameters (if any) of application components can be tuned. These adaptation levels are interrelated since they directly or indirectly impact system resource utilization and end-to-end QoS, which affects mission success. Adaptations at various levels must therefore be performed in a stable and *coordinated* fashion. In Chapter V we presented an *integrated* adaptive resource management architecture that performed system adaption at these levels in a coordinated fashion. However, in ultra large-scale systems [38], a single integrated resource management solution cannot be employed to manage the entire system, primarily due to scalability and reliability concerns. Therefore, future research is needed to design and develop techniques that enable the simultaneous operation of multiple resource management solutions in a coordinated and stable fashion.

- **Techniques that enable the certification of adaptive resource management solutions.** In the past, certification has been performed extensively in the domains of pharmaceuticals, health-care, automobile production, manufacturing, and assembly. Certification has not been widespread in the field of software development because the software industry is relatively young compared to other industries. However, recently since DRE systems are being used in many mission critical domains, certification or *verification and validation* [37] of such systems is gaining momentum. In order to certify the system that can be deployed in hostile environments, accurate *a priori* knowledge of the system behavior (system performance (QoS) and resource utilization) is required, and system behavior must meet the specified requirements. However, when adaptive resource management solutions are employed in a system, determining the behavior of the system *a priori* accurately is extremely difficult, if not impossible. Therefore, currently, the use of adaptive resource management solutions in mission critical DRE systems is minimal. Future research is necessary to study and develop new verification and validation techniques that enable the certification of adaptive resource management solutions, and thereby enabling the use of adaptive resource management solutions in mission critical systems.

APPENDIX A

LIST OF PUBLICATIONS

Our research on HiDRA and RACE has lead to the following journal, conference and workshop publications.

A.1 Refereed Journal Publications

1. Nishanth Shankaran, Nilabja Roy, Douglas C. Schmidt, Yingming Chen, Xenofon Koutsoukous, and Chenyang Lu, "The Design and Performance Evaluation of an Adaptive Resource-management Framework for Distributed Real-time Embedded Systems",*EURASIP Journal on Embedded Systems (EURASIP JES): Special Issue on Operating System Support for Embedded Real-Time Applications*, Edited by Michael Gonzalez, 2008.

2. Nishanth Shankaran, Xenofon Koutsoukos, Chenyang Lu, Douglas C. Schmidt, and Yuan Xue, "Hierarchical Control of Multiple Resources in Distributed Real-time and Embedded Systems", *the Springer Real-time Systems Journal*, Volume 39, Numbers 1-3, August, 2008, pages 237-282.

A.2 Refereed Conference Publications

1. Nilabja Roy, John S. Kinnebrew, Nishanth Shankaran, Gautam Biswas, and Douglas C. Schmidt, "Toward Effective Multi-capacity Resource Allocation in Distributed Real-time and Embedded Systems", *The 11th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing*, May 5-7 2007, Orlando, Florida.

2. Nishanth Shankaran, Douglas C. Schmidt, Yingming Chen, Xenofon Koutsoukous,

and Chenyang Lu, “The Design and Performance of Configurable Component Middleware for End-to-End Adaptation of Distributed Real-time Embedded Systems”, *The 10th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing*, May 7-9 2007, Santorini Island, Greece.

3. Amogh Kavimandan, Krishnakumar Balasubramanian, Nishanth Shankaran, Aniruddha Gokhale, and Douglas C. Schmidt, “QUICKER: A Model-driven QoS Mapping Tool”, *The 10th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing*, May 7-9 2007, Santorini Island, Greece.

4. John S. Kinnebrew, Ankit Gupta, Nishanth Shankaran, Gautam Biswas, and Douglas C. Schmidt, “A Decision-Theoretic Planner with Dynamic Component Reconfiguration for Distributed Real-Time Applications”, *The 8th International Symposium on Autonomous Decentralized Systems (ISADS 2007)*, Sedona, Arizona, Wednesday March 21 - Friday March 23, 2007.

5. Dipa Suri, Adam Howell, Douglas C. Schmidt, Gautam Biswas, John Kinnebrew, Will Otte, and Nishanth Shankaran, “A Multi-agent Architecture for Smart Sensing in the NASA Sensor Web”, *The 2007 IEEE Aerospace Conference*, Big Sky, Montana, March 3-10, 2007.

6. Nilabja Roy, Nishanth Shankaran, and Douglas C. Schmidt, “Bulls-Eye: A Resource Provisioning Service for Enterprise Distributed Real-time and Embedded Systems”, *Proceedings of the International Symposium on Distributed Objects and Applications (DOA)*, Montpellier, France, Oct 30 - Nov 1, 2006.

7. John Kinnebrew, Nishanth Shankaran, Gautam Biswas, and Douglas Schmidt, “A Decision-Theoretic Planner with Dynamic Component Reconfiguration for Distributed Real-Time Applications”, poster paper at the *Twenty-First National Conference on Artificial Intelligence*, Boston, Massachusetts, July 16-20, 2006.

8. Nishanth Shankaran, Xenofon Koutsoukos, Chenyang Lu, Douglas C. Schmidt, and Yuan Xue, "Hierarchical Control of Multiple Resources in Distributed Real-time and Embedded Systems", *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS 06),* Dresden, Germany, July 5-7, 2006.

9. Dipa Suri, Adam Howell, Nishanth Shankaran, John Kinnebrew, Will Otte, Douglas C. Schmidt, and Gautam Biswas, "Onboard Processing using the Adaptive Network Architecture", *Proceedings of the Sixth annual NASA Earth Science Technology Conference*, College Park, MD, June 27-29, 2006.

10. Nishanth Shankaran, Jaiganesh Balasubramanian, Douglas C. Schmidt, Gautam Biswas, Patrick Lardieri, Ed Mulholland, and Tom Damiano, "A Framework for (Re)Deploying Components in Distributed Real-time and Embedded Systems", poster paper at the *Dependable and Adaptive Distributed Systems, Track of the 21st ACM Symposium on Applied Computing*, Dijon, France, April 23-27, 2006.

11. Nishanth Shankaran, Raymond Klefsatd, "ZEUS: A CORBA Framework for Service Location and Creation", *Proceedings of the 2004 International Symposium on Applications and the Internet (SAINT)*, Tokyo Japan, January 26-30, 2004.

A.3 Refereed Workshop Publications

1. Nishanth Shankaran, John S. Kinnebrew, Xenofon D. Koutsoukos, Chenyang Lu, Douglas C. Schmidt, and Gautam Biswas, "Towards an Integrated Planning and Adaptive Resource Management Architecture for Distributed Real-time Embedded Systems", *Proceedings of the Workshop on Adaptive and Reconfigurable Embedded Systems (APRES)* at the *14th IEEE Real-Time and Embedded Technology and Applications Symposium*, St. Louis, MO, United States, April 22 - April 24, 2008.

2. John S. Kinnebrew, Nishanth Shankaran, Gautam Biswas, and Douglas C. Schmidt,

“A Decision-Theoretic Planner with Dynamic Component Reconfiguration for Distributed Real-time and Embedded Systems,” *Proceedings of the Workshop on Artificial Intelligence for Space Applications at IJCAI 2007*, Hyderabad, India, January 6-12, 2007.

3. John M. Slaby and Nishanth Shankaran, “Software Distribution in Ultra Large-scale Systems,” *Proceedings of the ACM OOPSLA 2006 Workshop on Ultra-Large-Scale Systems*, Portland, Oregon, October 26, 2006.

4. Nishanth Shankaran, Xenofon Koutsoukos, Douglas C. Schmidt, and Aniruddha Gokhale, “Evaluating Adaptive Resource Management for Distributed Real-Time Embedded Systems,” *Proceedings of the 4th Workshop on Adaptive and Reflective Middleware*, Grenoble, France, November 28, 2005.

REFERENCES

[1] *IEEE Std 802.11-1997 Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific requirements – Part 11: Wireless Lan Medium Access Control (MAC) And Physical Layer (PHY) Specifications*. IEEE Computer Society, 345 E. 47th St, New York, NY 10017, USA, Nov 1997.

[2] Tarek F. Abdelzaher, John Stankovic, Chengyang Lu, Ronghua Zhang, and Ying Lu. Feedback Performance Control in Software Services. *IEEE: Control Systems*, 23(3): 74–90, June 2003.

[3] Luca Abeni and Giorgio Buttazzo. Hierarchical QoS Management for Time Sensitive Applications. In *The Proceedings of the Seventh Real-Time Technology and Applications Symposium (RTAS)*, page 63, Washington, DC, USA, 2001. IEEE Computer Society.

[4] Ákos Lédeczi, Árpád Bakay, Miklós Maróti, Péter Völgyesi, Greg Nordstrom, Jonathan Sprinkle, and Gábor Karsai. Composing domain-specific design environments. *Computer*, 34(11):44–51, 2001. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/2.963443.

[5] Anne Thomas, Patricia Seybold Group. Enterprise JavaBeans Technology. java.sun.com/products/ejb/white_paper.html, December 1998. Prepared for Sun Microsystems, Inc.

[6] Karl Johan Astrom and Bjorn Wittenmark. *Computer-Controlled Systems: Theory and Design, Second Edition*. Prentice-Hall, Englewood Cliffs, NJ, 1990.

[7] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard real-time scheduling: The deadline monotonic approach. In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atalanta, 1991.

[8] Krishnakumar Balasubramanian, Jaiganesh Balasubramanian, Jeff Parsons, Aniruddha Gokhale, and Douglas C. Schmidt. A platform-independent component modeling language for distributed real-time and embedded systems. In *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 190–199, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2302-1. doi: http://dx.doi.org/10.1109/RTAS.2005.4.

[9] Giuseppe Bianchi. Performance Analysis of the IEEE 802.11 Distributed Coordination Function. *IEEE Journal on Selected Areas in Communications*, 18(1-2):535–547, Mar 2000. ISSN 0733-8716.

[10] Greg Bollella, James Gosling, Ben Brosgol, Peter Dibble, Steve Furr, David Hardin, and Mark Turnbull. *The Real-time Specification for Java*. Addison-Wesley, 2000.

[11] Stuart A. Boyer. *Supervisory Control and Data Acquisition*. ISA, 1993. ISBN 1556172109.

[12] S. Brandt, G. Nutt, T. Berk, and J. Mankovich. A Dynamic Quality of Service Middleware Agent for Mediating Application Resource Usage. In *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*, page 307, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-9212-X.

[13] Scott A. Brandt, Scott Banachowski, Caixue Lin, and Timothy Bisson. Dynamic Integrated Scheduling of Hard Real-Time, Soft Real-Time and Non-Real-Time Processes. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS '03)*, page 396, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2044-8.

[14] Owen Brown and Paul Eremenko. Fractionated Space Architectures: A Vision for Responsive Space. In *Proceedings of the 4th Responsive Space Conference*, Los Angeles, CA, 2006. American Institute of Aeronautics & Astronautics.

[15] Kevin Bryan, Lisa C. DiPippo, Victor Fay-Wolfe, Matthew Murphy, Jiangyin Zhang, Douglas Niehaus, David T. Fleeman, David W. Juedes, Chang Liu, Lonnie R. Welch, and Christopher D. Gill. Integrated CORBA Scheduling and Resource Management for Distributed Real-Time Embedded Systems. In *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 375–384, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2302-1. doi: dx.doi.org/10.1109/RTAS.2005.30.

[16] Bryan Buck and Jeffrey K. Hollingsworth. An API for Runtime Code Patching. *Int. J. High Perform. Comput. Appl.*, 14(4):317–329, 2000. ISSN 1094-3420. doi: dx.doi.org/10.1177/109434200001400404.

[17] Rolf Carlson. High-Security SCADA LDRD Final Report. Technical report, Advanced Information and Control Systems Department, Sandia National Laboratories, Albuquerque, New Mexico, USA, April 2002.

[18] Yingming Chen and Chenyang Lu. Flexible Maximum Urgency First Scheduling for Distributed Real-Time Systems. Technical Report WUCSE-2006-55, Washington University in St. Louis, October 2006.

[19] David Corman. WSOA-Weapon Systems Open Architecture Demonstration-Using Emerging Open System Architecture Standards to Enable Innovative Techniques for Time Critical Target (TCT) Prosecution. In *DASC'2001*, October 2001.

[20] David Corman, Jeanna Gossett, and Dennis Noll. Experiences in a Distributed Real-time Avionics Domain. In *Proceedings of the International Symposium on Object-Oriented Real-time Distributed Computing (ISORC)*, Washington, D.C., April 2002. IEEE/IFIP.

[21] CORPORATE Computer Science and Telecommunications Board. *Keeping the U.S. Computer Industry Competitive: Systems Integration*. National Academy Press, Washington, DC, USA, 1992. ISBN 0-309-04544-4.

[22] Tommaso Cucinotta, Luigi Palopoli, Luca Marzario, Giuseppe Lipari, and Luca Abeni. Adaptive Reservations in a Linux Environment. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 238–245, 2004.

[23] S. Curtis. The Magnetospheric Multiscale Mission...Resolving Fundamental Processes in Space Plasmas. *NASA STI/Recon Technical Report N*, pages 48257–+, December 1999.

[24] Dionisio de Niz and Raj Rajkumar. Partitioning Bin-Packing Algorithms for Distributed Real-time Systems. *International Journal of Embedded Systems*, 2(3):196–208, 2006.

[25] K.A. Delin and S.P. Jackson. Sensor Web for In Situ Exploration of Gaseous Biosignatures. 2000.

[26] Frank Dellaert and Chuck Thorpe. Robust Car Tracking Using Kalman Filtering and Bayesian Templates. In *Conference on Intelligent Transportation Systems*, 1997.

[27] Gan Deng, Jaiganesh Balasubramanian, William Otte, Douglas C. Schmidt, and Aniruddha Gokhale. DAnCE: A QoS-enabled Component Deployment and Configuration Engine. In *Proceedings of the 3rd Working Conference on Component Deployment (CD 2005)*, pages 67–82, Grenoble, France, November 2005.

[28] D. R. Fatland, M. J. Heavner, E. Hood, and C. Connor. The SEAMONSTER Sensor Web: Lessons and Opportunities after One Year. *AGU Fall Meeting Abstracts*, pages A3+, December 2007.

[29] John D. Fernandez and Andres E. Fernandez. SCADA Systems: Vulnerabilities and Remediation. *J. Comput. Small Coll.*, 20(4):160–168, 2005.

[30] David Fleeman, Matthew Gillen, A. Lenharth, M. Delaney, Lonnie R. Welch, David W. Juedes, and Chang Liu. Quality-Based Adaptive Resource Management Architecture (QARMA): A CORBA Resource Management Service. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*. IEEE Computer Society, 2004.

[31] G. F. Franklin, J. D. Powell, and M. Workman. *Digital Control of Dynamic Systems, 3rd edition*. Addition-Wesley, 1997.

[32] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[33] Christopher D. Gill. *Flexible Scheduling in Middleware for Distributed Rate-Based Real-time Applications*. PhD thesis, Department of Computer Science, Washington University, St. Louis, 2002.

[34] M. González Harbour, J. J. Gutiérrez García, J. C. Palencia Gutiérrez, and J. M. Drake Moyano. MAST: Modeling and Analysis Suite for Real Time Applications. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems (ECRTS '01)*, page 125, Washington, DC, USA, 2001. IEEE Computer Society.

[35] John Hatcliff, William Deng, Matthew Dwyer, Georg Jung, and Venkatesh Prasad. Cadena: An Integrated Development, Analysis, and Verification Environment for Component-based Systems. In *Proceedings of the 25th International Conference on Software Engineering*, pages 160–172, Portland, OR, May 2003.

[36] Gavin Holland, Nitin Vaidya, and Paramvir Bahl. A Rate-Adaptive MAC Protocol for Multi-Hop Wireless Networks. In *MobiCom '01: Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pages 236–251, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-422-3. doi: doi.acm.org/10.1145/381677.381700.

[37] IEEE Computer Society. *Std-1012 1998: IEEE Standard for Software Verification and Validation*. New York, 1998.

[38] Software Engineering Institute. Ultra-Large-Scale Systems: Software Challenge of the Future. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, Jun 2006.

[39] John Kinnebrew, Nishanth Shankaran, Gautam Biswas, and Douglas Schmidt. A Decision-Theoretic Planner with Dynamic Component Reconguration for Distributed Real-Time Applications. In *Poster paper at the Twenty-First National Conference on Artificial Intelligence*, Boston, MA, July 2006.

[40] Sharath Kodase, Shige Wang, Zonghua Gu, and Kang G. Shin. Improving Scalability of Task Allocation and Scheduling in Large Distributed Real-time Systems using Shared Buffers. In *Proceedings of the 9th Real-time/Embedded Technology and Applications Symposium (RTAS 2003)*, Washington, DC, May 2003. IEEE.

[41] Xenofon Koutsoukos, Radhika Tekumalla, Balachandran Natarajan, and Chenyang Lu. Hybrid Supervisory Control of Real-time Systems. In *IEEE Real-time and Embedded Technology and Applications Symposium*, San Francisco, California, March 2005. IEEE Computer Society.

[42] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *RTSS '89: Proceedings of the IEEE Real-Time Systems Symposium*, pages 166–171, Washington, DC, USA, 1989. IEEE Computer Society. doi: 10.1109/REAL.1989.63567.

[43] Baochun Li and Klara Nahrstedt. A Control-based Middleware Framework for QoS Adaptations. *IEEE Journal on Selected Areas in Communications*, 17(9):1632–1650, September 1999.

[44] Giuseppe Lipari, Gerardo Lamastra, and Luca Abeni. Task Synchronization in Reservation-Based Real-Time Systems. *IEEE Trans. Computers*, 53(12):1591–1601, 2004.

[45] C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-time Environment. *JACM*, 20(1):46–61, January 1973.

[46] Jane W. S. Liu. *Real-time Systems*. Prentice Hall, New Jersey, 2000.

[47] Jane W.S. Liu, Juan Redondo, Zhong Deng, Too Tia, Riccardo Bettati, Ami Silberman, Matthew Storch, Rhan Ha, and Wei Shih. PERTS: A Prototyping Environment for Real-Time Systems. Technical report, Champaign, IL, USA, 1993.

[48] M. Lopez S., S. Armando Alfonzo G., J. Perez O., J.G. Gonzalez S., and A. Montes R. A metamodel to carry out reverse engineering of c++ code into uml sequence diagrams. *Electronics, Robotics and Automotive Mechanics Conference, 2006*, 2:331–336, Sept. 2006. doi: 10.1109/CERMA.2006.100.

[49] Joseph P. Loyall, Richard E. Schantz, David Corman, James L. Paunicka, and Sylvester Fernandez. A Distributed Real-Time Embedded Application for Surveillance, Detection, and Tracking of Time Critical Targets. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 88–97, San Francisco, CA, 2005.

[50] Chenyang Lu, John A. Stankovic, Sang H. Son, and Gang Tao. Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms. *Real-Time Syst.*, 23 (1-2):85–126, 2002.

[51] Chenyang Lu, Xiaorui Wang, and Christopher Gill. Feedback Control Real-time Scheduling in ORB Middleware. In *Proceedings of the 9th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS)*, pages 37–48, Washington, DC, May 2003.

[52] Chenyang Lu, Xiaorui Wang, and Xenofon Koutsoukos. Feedback Utilization Control in Distributed Real-time Systems with End-to-End Tasks. *IEEE Trans. on Par. and Dist. Sys.*, 16(6):550–561, 2005. ISSN 1045-9219. doi: dx.doi.org/10.1109/TPDS.2005.73.

[53] Prakash Manghwani, Joseph Loyall, Praveen Sharma, Matthew Gillen, and Jianming Ye. End-to-End Quality of Service Management for Distributed Real-Time Embedded Applications. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2005)*, volume 03, Los Alamitos, CA, USA, 2005.

[54] Pau Marti, Caixue Lin, Scott A. Brandt, Manel Velasco, and Josep M. Fuertes. Optimal State Feedback Based Resource Allocation for Resource-Constrained Control Tasks. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*, pages 161–172, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2247-5. doi: dx.doi.org/10.1109/REAL.2004.39.

[55] D. Mills. The Network Time Protocol. In *RFC 1059*. Network Working Group, 1988.

[56] Ingo Molnar. Linux with Real-time Pre-emption Patches. http://www.kernel.org/pub/linux/kernel/projects/rt/, Sep 2006.

[57] *Light Weight CORBA Component Model Revised Submission*. Object Management Group, OMG Document realtime/03-05-05 edition, May 2003.

[58] Object Management Group. *The Common Object Request Broker: Architecture and Specification, Revision 2.6*. Object Management Group, December 2001.

[59] *Common Object Request Broker Architecture Version 1.3*. Object Management Group, OMG Document formal/2004-03-12 edition, March 2004.

[60] *CORBA Components*. Object Management Group, OMG Document formal/2002-06-65 edition, June 2002.

[61] *Deployment and Configuration Adopted Submission*. Object Management Group, OMG Document mars/03-05-08 edition, July 2003.

[62] Object Management Group. *Real-time CORBA Specification*. Object Management Group, OMG Document formal/05-01-04 edition, August 2002.

[63] Moonju Park and Yookun Cho. Feasibility Analysis of Hard Real-Time Periodic Tasks. *J. Syst. Softw.*, 73(1):89–100, 2004. ISSN 0164-1212. doi: dx.doi.org/10.1016/S0164-1212(02)00236-X.

[64] Binoy Ravindran, Lonnie Welch, and Behrooz Shirazi. Resource Management Middleware for Dynamic, Dependable Real-Time Systems. *Real-Time Syst.*, 20(2):183–196, 2001. ISSN 0922-6443. doi: dx.doi.org/10.1023/A:1008141921230.

[65] Richard Schantz, Joseph Loyall, Michael Atighetchi, and Partha Pal. Packaging Quality of Service Control Behaviors for Reuse. In *Proceedings of the* 5^{th} *IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC)*, pages 375–385, Crystal City, VA, April/May 2002.

[66] Douglas C. Schmidt and Stephen D. Huston. *C++ Network Programming, Volume 2: Systematic Reuse with ACE and Frameworks*. Addison-Wesley, Reading, Massachusetts, 2002.

[67] Douglas C. Schmidt, David L. Levine, and Sumedh Mungee. The Design and Performance of Real-time Object Request Brokers. *Computer Communications*, 21(4): 294–324, April 1998.

[68] Douglas C. Schmidt, Rick Schantz, Mike Masters, Joseph Cross, David Sharp, and Lou DiPalma. Towards Adaptive and Reflective Middleware for Network-Centric Combat Systems. In *CrossTalk - The Journal of Defense Software Engineering*, pages 10–16, Hill AFB, Utah, USA, nov 2001. Software Technology Support Center.

[69] Samarth H. Shah, Kai Chen, and Klara Nahrstedt. Dynamic Bandwidth Management for Single-hop Ad Hoc Wireless Networks. *Mob. Netw. Appl.*, 10(1-2):199–217, 2005. ISSN 1383-469X. doi: doi.acm.org/10.1145/1046430.1046445.

[70] Nishanth Shankaran, Xenofon Koutsoukos, Chenyang Lu, Douglas C. Schmidt, and Yuan Xue. Hierarchical Control of Multiple Resources in Distributed Real-time and Embedded Systems. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS 06)*, Dresden, Germany, July 2006.

[71] Praveen Kaushik Sharma, Joseph P. Loyall, George T. Heineman, Richard E. Schantz, Richard Shapiro, and Gary Duzan. Component-based dynamic qos adaptations in distributed real-time and embedded systems. In *CoopIS/DOA/ODBASE (2)*, pages 1208–1224, Agia Napa, Cyprus, 2004. Springer.

[72] David C. Sharp and Wendy C. Roll. Model-Based Integration of Reusable Component-Based Avionics System. Proceedings of the Workshop on Model-Driven Embedded Systems in RTAS 2003, May 2003.

[73] David C. Sharp, Edward Pla, Kenn R. Luecke, and Ricardo J. Hassan II. Evaluating Real-time Java for Mission-Critical Large-Scale Embedded Systems. In *IEEE Real-time and Embedded Technology and Applications Symposium*, Washington, DC, May 2003. IEEE Computer Society.

[74] John A. Stankovic, Ruiqing Zhu, Ram Poornalingam, Chenyang Lu, Zhendong Yu, Marty Humphrey, and Brian Ellis. VEST: An Aspect-Based Composition Tool for Real-Time Systems. In *RTAS '03: Proceedings of the The 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, page 58, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1956-3.

[75] David B. Stewart and Pradeep K. Khosla. Real-time Scheduling of Sensor-Based Control Systems. In W. Halang and K. Ramamritham, editors, *Real-time Programming*. Pergamon Press, Tarrytown, NY, 1992.

[76] SUN. Java Remote Method Invocation (RMI) Specification. `java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html`, 2002.

[77] Sun Microsystems. Enterprise JavaBeans Specification. java.sun.com/products/ejb/docs.html, August 2001.

[78] Dipa Suri, Adam Howell, Nishanth Shankaran, John Kinnebrew, Will Otte, Douglas C. Schmidt, and Gautam Biswas. Onboard Processing using the Adaptive Network Architecture. In *Proceedings of the Sixth Annual NASA Earth Science Technology Conference*, College Park, MD, June 2006.

[79] G. K. Wallace. The JPEG Still Image Compression Standard. *Communications of the ACM*, 34(4):30–44, April 1991.

[80] Nanbor Wang and Christopher Gill. Improving real-time system configuration via a qos-aware corba component model. In *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9*, page 90273.2, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2056-1.

[81] Nanbor Wang, Douglas C. Schmidt, Aniruddha Gokhale, Craig Rodrigues, Balachandran Natarajan, Joseph P. Loyall, Richard E. Schantz, and Christopher D. Gill. QoS-enabled Middleware. In Qusay Mahmoud, editor, *Middleware for Communications*, pages 131–162. Wiley and Sons, New York, 2004.

[82] Xiaorui Wang, HuangMing Huang, Venkita Subramonian, Chenyang Lu, and Christopher Gill. CAMRIT: Control-based Adaptive Middleware for Real-time Image Transmission. In *Proc. of the 10th IEEE Real-time and Embedded Tech. and Applications Symp. (RTAS)*, Toronto, Canada, May 2004.

[83] Xiaorui Wang, Dong Jia, Chenyang Lu, and Xenofon Koutsoukos. Decentralized utilization control in distributed real-time systems. In *RTSS '05: Proceedings of the 26th IEEE International Real-Time Systems Symposium*, pages 133–142, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2490-7. doi: dx.doi.org/10.1109/RTSS.2005.15.

[84] Xiaorui Wang, Chenyang Lu, and Xenofon Koutsoukos. Enhancing the Robustness of Distributed Real-Time Middleware via End-to-End Utilization Control. In *RTSS '05: Proceedings of the 26th IEEE International Real-Time Systems Symposium*, pages 189–199, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2490-7. doi: dx.doi.org/10.1109/RTSS.2005.20.

[85] Greg Welch and Gary Bishop. An introduction to the Kalman Filter: Course 8. In *Computer Graphics, Annual Conference on Computer Graphics and Interactive Techniques*, Los Angeles, CA, USA, August 2001. SIGGRAPH, ACM Press, Addison-Wesley Publishing Company.

[86] L. R. Welch, B. A. Shirazi, B. Ravindran, and C. Bruggeman. DeSiDeRaTa: QoS Management Technology for Dynamic, Scalable, Dependable Real-time Systems. In

IFACs 15th Workshop on Distributed Computer Control Systems (DCCS98). IFAC, September 1998.

[87] Brian White and Jay Lepreau et al. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002. USENIX Association.

[88] Victor Fay Wolfe, Lisa C. DiPippo, Ramachandra Bethmagalkar, Gregory Cooper, Russell Johnston, Peter Kortmann, Ben Watson, and Steven Wohlever. RapidSched: Static Scheduling and Analysis for Real-Time CORBA. In *WORDS '99: Proceedings of the Fourth International Workshop on Object-Oriented Real-Time Dependable Systems*, page 34, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0101-X.

[89] Ann Wollrath, Roger Riggs, and Jim Waldo. A Distributed Object Model for the Java System. *USENIX Computing Systems*, 9(4):265–290, November/December 1996.

[90] John A. Zinky, David E. Bakken, and Richard Schantz. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, 3(1): 1–20, 1997.