Scalable, Distributed, Dynamic Resource Management for the ARMS Distributed Real-Time Embedded System

Kurt Rohloff, Yarom Gabay, Jianming Ye and Richard Schantz
BBN Technologies
Cambridge, MA, 02138 USA
{krohloff, ygabay, jye, schantz}@bbn.com

Abstract

We present a scalable, hierarchical control system for the dynamic resource management of a distributed real-time embedded (DRE) system. This DRE is inspired by the DARPA Adaptive and Reflective Middleware Systems (ARMS) program. The goal of the control system is to simultaneously manage multiple resources and QoS concerns using a utility-driven approach for decision making and performance evaluation. At each level of the control hierarchy there are multiple local controllers which autonomously make decisions to optimize their local utility. The controllers in the hierarchy can use different, localized resource control algorithms and the system's user can tune the operations of the local controllers. We discuss how the selections of local control algorithms affect the behavior of the overall system. The control system is designed to be easily adaptable to other multi-tiered DRE systems.

1. Introduction

Large distributed real-time embedded systems are often designed with static resource management strategies tailored for specific goals or missions. These rigid resource allocation strategies are incapable of adapting to changing system goals, resource levels and operating environments. This inability to adapt can cause DRE systems to fail to meet end-to-end quality of service (QoS) requirements when conditions change.

We present a hierarchical control system for the dynamic resource management of hierarchical DRE systems that is capable of simultaneously managing multiple resources and QoS concerns. Dynamic resource management has the capability to achieve much higher performance in a constrained resource system than static resource management approaches.

The DRE application area is inspired by the DARPA Adaptive and Reflective Middleware Systems (ARMS) program in conjunction with Raytheon and Lockheed Martin. The ARMS system can be decomposed into multiple missions and missions can be decomposed into multiple submissions called application strings or strings. A multi-tiered behavioral hierarchy such as this is a common aspect of many DRE systems.

A key element of the control system we are designing for these DRE systems is a utility-driven approach for decision-making and performance evaluation with respect to resource allocation in the controllers. Utility is computed for each element in the system hierarchy (string, mission, system) and is a measure of that element's ability to perform its desired tasks. The allocation of system resources is dynamically managed to locally maximize utility at each level of the system hierarchy with individual controllers deployed for the whole system and all missions and its strings.

The general philosophy for the control system is a bottom-up approach to dynamic resource management. At the lowest levels, controllers perform fast, frequent, local tunings of system behavior, while at the highest levels, controllers perform less frequent, but more aggressive control actions. This bottom-up scalable control approach can be easily applied to other hierarchical DRE systems. Other

1-4244-0910-1/07/$20.00 ©IEEE

dynamic control approaches to meet QoS requirements are in [2, 3].

The next section describes key elements of the multi-tiered hierarchical system we are controlling. The control objectives and utility measures for dynamic resource management are outlined in Section 3. The control architecture and the algorithms used by the controllers are discussed in Section 4. We discuss the behavioral effects of using the various control algorithms in Section 5. We conclude the paper and discuss several avenues of future work in Section 6.

2. System Architecture

Properties of DRE systems can be understood via aspects of both their resources and applications running on those resources. The resource aspects of DRE systems include the computation and communication resources of the system. The computational resources are a set of general purpose computer hosts. The communication resources in the system are the communication links formed between various hosts in the systems and the attributes of these links such as bandwidth, maximum delay and operating modes.

Hosts are assumed to be grouped into pools or clusters of computing resources based on their physical locations. Pools are managed independently of one another by local pool managers. Communication between hosts in the same pool is assumed to be generally inexpensive, while hosts in a pool share limited communication gateways to hosts in other pools. Therefore, communications between hosts is partitioned into intra- and inter-pool communications. A diagram of the system resource interactions can be seen in Figure 1.

Software applications are deployed onto the computational resources and can be viewed at multiple levels of abstraction. At the lowest level of abstraction, applications run on hosts and perform work requiring certain computing resources.

At the next highest level of abstraction, an application string, or string, is a logical sequence of applications that sequentially process information with unique starting and ending applications. Strings are generally deployed across multiple hosts and pools, so string controllers need to manage inter- and intra-pool communication. Strings generally perform work subject to end-to-endQoS requirements. Two or more strings may share an application.

At the penultimate level of abstraction, a mission is a group of strings that cooperate to achieve common goals. At the highest level of abstraction, the system incorporates all running missions and resources those missions have access to. A schematic of the system-mission-string decomposition can be seen in Figure 2.

The ARMS system has software components called the Infrastructure Allocator (IA) that allocates applications to hosts and a Bandwidth Broker (BB) that allocates bandwidth on intra- and inter-pool communication links. The IA actuates the control actions of the control system by (re)allocating computational resources, and the BB actuates the control actions of the control system by (re)allocating communication resources. It is assumed that the DRE systems considered in this paper have similar software components to actuate control actions. It is also assumed that system status measurement information is shared by a distributed publish-and-subscribe service called RSS (Resource Status Service). RSS allows system components to improve efficiency by not

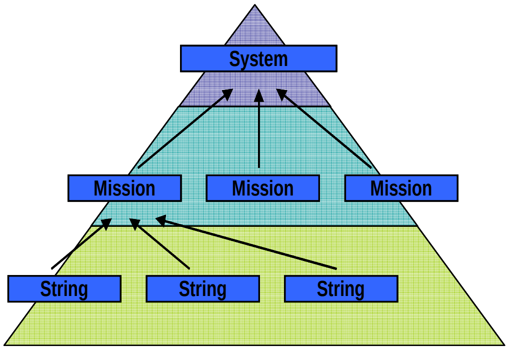


Figure 2: Software Hierarchy of DRE Systems

having to redundantly gather and distribute status information independently([4]).

3. Resource Management Objectives

The hierarchical control system uses a set of utility functions to evaluate the performance of strings and missions in the system against current resource allocations. The control system also uses the utility estimation function to estimate the desirability of various control actions with respect to the future performance and utility of the system. The control system chooses control actions that would result in a higher level of estimated utility.

If the system has enough unused system resources, the system could allocate resources to previously undeployed missions or application strings to boost its overall utility and performance. Conversely, if resource contention were to occur due to an over-deployment of missions (possibly due to resource failure among other possible causes), then the performance and utility of the deployed missions would drop. A change in resource availability indicates that the controllers may need to adjust resource usage to attempt to maximize utility due to the current operating conditions.

We use a set of hierarchical utility functions to measure the performance of the system that follows the system-mission-string hierarchy outlined in the introduction. Utility functions are defined for the system, each of the missions and each of the application strings that measure the performance of these entities under current resource allocation. More formally, at a given time t:

$U(t)$ is the utility of system performance.

$U_i^m(t)$ is the utility of mission i.

$U_i^{s_j}(t)$ is the utility of string j of mission i.

We define the system-level utility, $U(t)$ to be a weighted sum of the mission-level utilities:

$$U(t) = \sum_{i=0}^{M} w_i^m U_i^m(t)$$

The weight factor w_i^m is a measure of the relative importance of mission i.

Similarly, the mission's ability to complete its required tasks depends on the ability of its strings to complete their desired tasks, so mission utility $U_i^m(t)$ is a weighted sum of the mission's string-level utilities:

$$U_i^m(t) = \sum_{j=0}^{S_i} w_i^{S_j} U_i^{s_j}(t)$$

The weight factor $w_i^{S_j}$ is a measure of the relative importance of string s_j of mission i.

The utility of string s_j from mission i depends on the timeliness, quality, and throughput of information processed by the string. These factors are an indication of how well the string can process and transmit information. Timeliness ($T_i^{s_j}(t)$) is a measure of the application string's ability to meet end-to-end real-time requirements. Quality ($q_i^{s_j}(t)$) is a measure of how useful the information processed by an application string is. Throughput ($Th_i^{s_j}(t)$) is the rate at which information to be processed is sent to the string. The mapping of the $T_i^{s_j}(t)$, $q_i^{s_j}(t)$ and $Th_i^{s_j}(t)$ to the utility of the string may vary from string to string, so we define $U_i^{s_j}(t)$ to be computed by a generic function $F_i^{s_j}(\cdot,\cdot,\cdot)$:

$$U_i^{s_j}(t) = F_i^{s_j}\left(T_i^{s_j}(t), q_i^{s_j}(t), Th_i^{s_j}(t)\right).$$

The utility $U_i^{s_j}(t)$ is periodically computed by its string controller and published on RSS so that the higher level controllers can compute $U_i^m(t)$ and $U(t)$. Expressions for timeliness, quality, and throughput are application dependent.

4. Control Architecture

In order to hierarchically allocate resources in the system to maximize system utility, controllers are deployed with one controller for every string (called the string controllers), one controller for every mission (called the Mission Controllers or MC's) and one system controller (called the Multi-Mission Coordinator or MMC). A diagram of the system-mission-string hierarchy can be seen in Figure 3. At the top of the diagram, the MMC controls the gross allocation of resources to the missions. At the next level down, the local MC's coordinate the local deployment of strings which consume the local allocation of resources. At the lowest level, the string controllers, fast local tunings of the local resource usages.

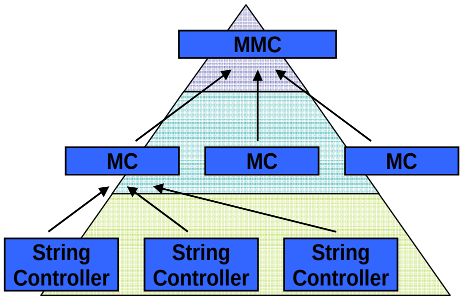


Figure 3: Control Hierarchy of the DRE System

All of the controllers in the hierarchy communicate with their parents and/or children to facilitate tradeoffs between local run-time utilities and resource allocations among control layers in the bottom-up control design. The controllers interact with each other through direct communications, but the controllers receive information about system resource status or performance through RSS.

The low level controllers are generally fast and responsive, while the high level controllers have the ability to take more aggressive control actions. Higher level control actions are more invasive, so the higher level controllers are designed take more time to better estimate which of their control actions will maximize their local utility. Local controllers in this design attempt to greedily maintain their local utility and the bottom-up control philosophy limits local, fast utility gains that are potentially detrimental to the overall system utility.

5. Local Control Algorithms

We now discuss the operations of the individual controllers at each level of the control hierarchy.

5.1 String Controller

String controllers perform fast low-level tuning of quality and throughput in order to maintain their local string utility. A drop in string utility could be caused by either resource contention or failure, but on resource failures, the string controller is expected to receive notification about the failures from RSS. In the absence of a notification from RSS indicating otherwise, the string controller assumes drops in utility are caused by resource contention.

Generally the quality ($q_i^{s_j}(t)$) and the throughput ($Th_i^{s_j}(t)$) of an application string can be directly controlled by the string's controller by adjusting applications in a string, but the timeliness ($T_i^{s_j}(t)$) cannot. However, the timeliness of a string can be influenced by tuning the quality and throughput of information processed by a string. When a string controller observes that $U_i^{s_j}(t)$ is significantly below its measured baseline, the string controller attempts to decrease the string's quality and throughput. Any observed improvement in timeliness by decreasing quality and throughput will not be instantaneous, so incremental decreases are made in both quality and throughput on the utility measurement cycles. Quality and throughput are continually decremented until a local maximum of the measured string utility $U_i^{s_j}(t)$ is found.

If the local maximum is not sufficiently close to the utility baseline, the string controller sends a signal to the mission controller that the mission controller should attempt to relieve the string's observed resource contention. It remains an open problem to determine how aggressively the string controllers should decrement quality and throughput in attempts to maintain their local utility.

5.2 Mission Controller

When given access to an amount of resources, an MC decides which of its mission's strings should be deployed using those resources to maximize the mission's utility. We have designed an ARMS mission controller that operates with two algorithmic components. A schematic of the mission controllers' internal operational logic is seen in Figure 4. One mission control algorithmic component, called the *string selection logic*, determines which strings to deploy/kill/redeploy based on the strings' importance to the mission, the amount of resources the mission is allowed to use and the strings' current deployment status. The second algorithmic component, called the string binding logic, selects the resources that deployed strings should use based on how much resources are available to the mission. The string binding logic interfaces with the IA to determine which resources are free and strings should be deployed on.

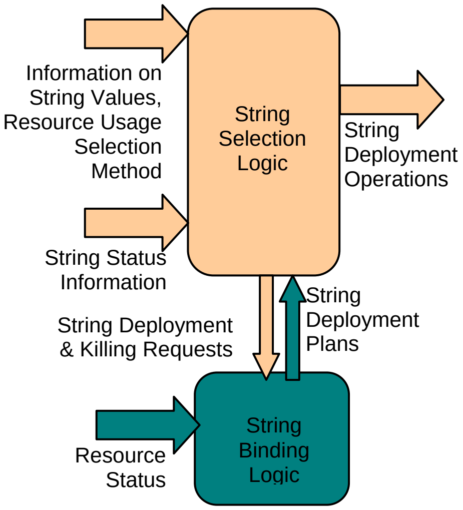

Information on String Values, Resource Usage Selection Method
String Selection Logic
String Deployment Operations
String Status Information
String Deployment & Killing Requests
String Deployment Plans
Resource Status
String Binding Logic

Figure 4: Mission Controller Logic Schematic

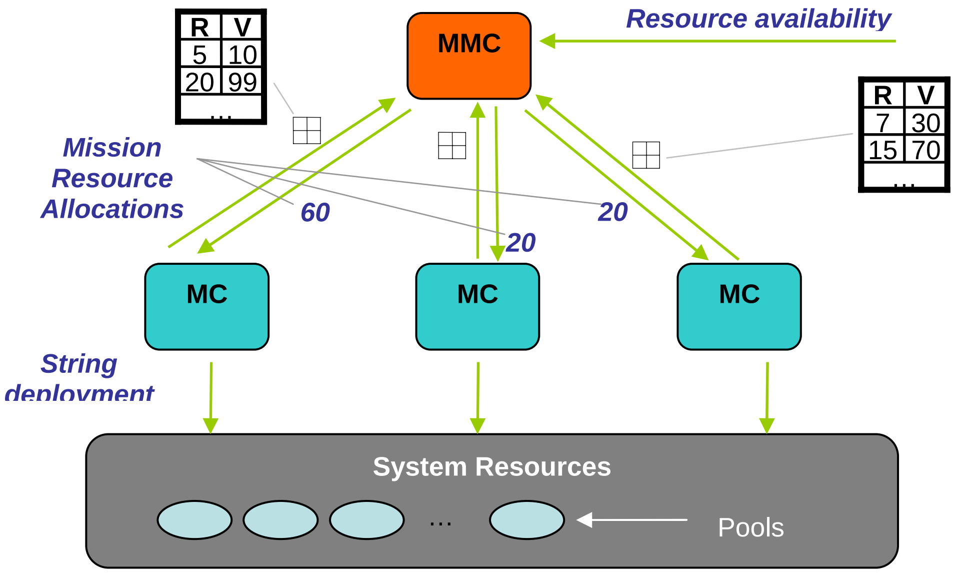
The string selection logic operates in response to partial system failures and user input to ensure that importance revaluation are followed through the missions' string deployments. We have tested various algorithms for the string selection logic. These algorithms include importance-based greedy ordering, resource-efficiency-based greedy ordering and dynamic programming.

When selecting which strings to deploy, the amount of resources the mission is allowed to use is intended to be used as an input from the MMC so that the MMC can direct the overall division of resources to the missions. Note that instead of being allocating specific resources, the missions are given input as to how much resources they are allowed to use.

5.3 Multi-Mission Coordinator

The MMC performs the gross-level allocation of resources between the missions. Note that rather than giving the mission controllers access to specific resources, the MMC gives the mission controllers the right to use an *amount* of resources.

When dividing the available system resources up amongst the missions, the MMC predicts what utility the system would attain from allocating various amounts of resources to the missions. To do this, the MMC receives a lookup tables from each mission controllers that maps an approximation of the sums of the importance values of strings the mission controllers could deploy for their missions if given the ability to use various levels of resources. The lookup tables are generated and sent to the MMC by every mission controller at initialization and are based on user-commanded mission goals. The lookup tables are also intended to be updated regularly whenever a mission receives a command directive to refine its local behavior based on the relative importance values of the missions and its strings. Figure 5 contains a schematic of MMC operation which indicates that the lookup tables of



R	V
5	10
20	99
…	

R	V
7	30
15	70
…	

Figure 5: MMC Concept of Operations

missions' resource-value mappings.

When the lookup tables are sent to the mission controller, the resource levels listed in the lookup table are quantized based on the levels of quality of service provided by the missions for deploying groups of strings. Because the deployment of the missions' most important/critical strings is necessary for minimal mission operation, the lowest quantization level of the lookup tables correspond to the resources necessary to deploy just the most important/critical strings for each of the missions. The other quantization levels in the mission lookup tables are dependent upon the context of the mission and could be used to tune the operation of the MMC when dividing system resources among the missions.

When the MMC has the lookup tables from the mission controllers and given information about the availability of resources in the system, the MMC needs to decide how much resources should be provided to every mission controller in order to guarantee all critical strings can be run and to maximize the total value of all strings that can be deployed. The problem of the MMC allocating resources to the missions can be formalized as a multiple-choice knapsack problem [1].

The MMC could use any number of algorithms to compute the most efficient division of resources based on information from the lookup tables and resource efficiency. We found the dynamic programming algorithm to be very effective and efficient considering the relatively small numbers of missions that we are using.

Once the MMC computes the division of the resources amongst the missions, the MMC communicates to the mission controllers how much of the computation resources they are each allowed to use. This communication is indicated in the schematic of the MMC operation in Figure 5.

When the mission controllers have information about how many resources they are allowed to use, they decide which of their strings to deploy in order to maximize local utility. The mission controllers receive no information about the allocation of resources to other missions. Therefore, on the occurrence of significant system events such as partial system failures or impotance revaluation, the mission controllers make all of their local resource allocation decisions under the assumption that their total resource allocation hasn't changed unless they receive updated information from the MMC.

6. System Simulation

We developed a large-scale, highly configurable Matlab/Simulink model of the ARMS multi-mission system to objectively compare the utility-measured performance of the system using the dynamic resource controller to a baseline system where resources are statically allocated at initialization.

For our simulation experiments, we configured the model to consist of three missions with 100 strings each that can be deployed on 5 pools with inter-pool link resources. When the mission controllers perform string deployment operations, there is a configurable actuation delay between the time the mission controller sends the actuation signal until the time the string becomes operational which we approximated as 0.1sec.

In the simulation model, the operating conditions of the strings are highly configurable. The computational and communication requirements of the strings can be customized to model various mission scenarios as long as there are at least two applications in every string. The user-assigned importance values of the strings are also configurable and can be used as experimental parameters in simulation.

The amount of resources available to the multi-mission system can also be adjusted in the simulation model. In particular, the number of pools and how many applications can be run in each pool and be individually adjusted along with the amount of inter-pool bandwidth available to the mission's strings in the system's inter-pool communication links. It is not necessary that the pools and links have homogenous resource configurations. We simulate partial system failures in real-time in the model by removing all of a pool's nodes to model a complete pool failures.

Using the large-scale Matlab/Simulink model of the ARMS system, we generated 100 experimental string deployment scenarios consisting of 3 missions of 100 strings, each with randomly chosen application lengths uniformly distributed between 2 and 11. Inter-application bandwidth requirements were randomly chosen to be either 1 or 2 megabits per second. The 100 strings were randomly assigned integer importance values with a uniform random distribution between 1 and 10, inclusive. To generate the lookup tables generated by the mission controllers and sent to the MMC, we randomly grouped the missions' string sets into 10 quantization levels.

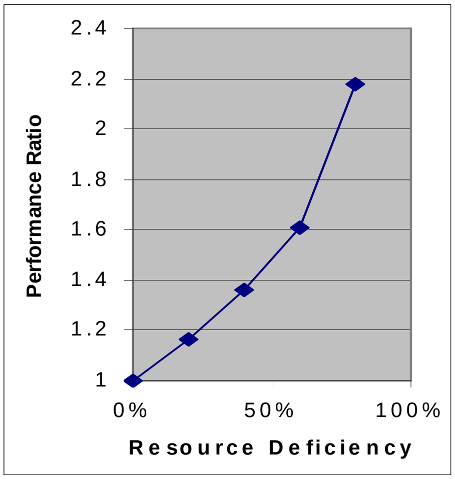


Figure 6: Ratio of Dynamic to Static Utility Performance vs. Resource Deficiency

For each scenario, the system had five operational pools at initialization with sufficient resources deploy all strings. The pools were allocated computation resources such that after the failure of a specific pool, the mission controller would cause the mission to have only 80% of the resources required to deploy all strings. The failure of two pools would cause the system to have 60% of the resource to deploy all strings, the failure of three pools would cause the system to have 40% of the resources to deploy all strings and the failure of four pools would cause the system to have 20% of the resource to deploy all strings.

The Matlab/Simulink simulations were run such that the MMC and mission controllers were given sufficient time to deploy all strings after initialization. After initialization was completed, a set of pools was failed, and the mission controller was allowed to complete its recovery operations in response to the pool failure. We recorded the utility attained by the mission controller immediately before the failure and after failure recovery operations completed.

We also collected data on the performance of the system if a dynamic resource controller was used where resources are allocated at initialization and then no changes are made in the resource allocation. This static resource allocation strategy was the baseline system in the ARMS program.

Figure 6 contains a graph that demonstrates how the ratio of performance for systems using the static and dynamic MMC's vary with resource deficiency. As can be seen from the graph, as resource deficiency increases, the dynamic MMC is able to achieve over 2x performance gains over the static MMC.

7. Conclusions

We have presented a utility driven hierarchical controller design for multi-tiered DRE systems to dynamically manage system resources. Although only three levels of abstraction are considered here for the DRE and control systems, our design is easily scaled to any number of control levels where low level controllers take fast, limited actions and high level controllers take slow, more invasive actions.

Acknowledgements

This work was supported by the Defense Research Projects Agency (DARPA) under contract NBCHC030119. Approved for public release. Distribution unlimited.

Bibliography

[1] K. Dudziński and S. Walukiewicz. "Exact methods for the knapsack problem and its generalizations", European Journal of Operations Research, 28, pg 3-21, 1987

[2] C. Lu, J. Stankovic, S. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, 23(1–2):85–126, July 2002.

[3] H. Wu, B. Ravindran, E. Jensen, and P. Li. Time/utility function decomposition techniques for utility accrual scheduling algorithms in real-time distributed systems. *IEEE Transactions on Computers*, 54(9):1138–1153, 2005.

[4] J. Zinky, J. Loyall, and R. Schapiro. Runtime performance modeling and measurement of adaptive distributed object applications. In *Proceeding of International Symposium on Distributed Object and Applications (DOA)*, Irvine, CA, 2002.