

# Quality Measures for Embedded Systems and Their Application to Control and Certification\*

Kurt Rohloff, Joseph Loyall, Richard Schantz

BBN Technologies

10 Moulton St. Cambridge, MA 02138

{krohloff, jloyall, schantz}@bbn.com

## Abstract:

Distributed, real-time embedded (DRE) systems, such as that being developed under the DARPA Adaptive Reflective Middleware System (ARMS) program, require predictable, controlled real-time behavior. It is a challenge to develop ways to measure and evaluate the quality, or *utility*, of DRE system performance in the context of the dynamic, unpredictable environments in which they operate. This difficulty runs hand-in-hand with the challenge of providing adequate *control* of the system to effect behavior in the system that will allow the controlled system to be *certified* as exhibiting correct behavior. As part of the ARMS program, we have been developing utility-based measures of system quality for *assessing* correct behavior of a system and for *driving* feedback control at runtime. This paper describes the real-time utility measures we have developed for ARMS and their use as feedback signals to a control system that manages the dynamic allocation of resources to applications in the system. We then explore the issues associated with using utility-based assessment as part of an evidence-based certification process for dynamic real-time systems.

## 1. Introduction

The goal of certification is to document (to the satisfaction of some certification authority) that correct system behavior will occur in the situations and environments the system will be fielded. Traditionally, certification has involved a combination of adherence to documented processes, testing, and formal analysis. Traditional testing and formal analysis methods face difficulties when applied to dynamic systems as it is more difficult to identify a discrete set of states (and therefore a finite set of tests) that would cover the whole spectrum of possible operations of the system. Additionally, there is a richer set of inputs (including environment conditions and nondeterministic decisions) that affect dynamic system behavior and can be difficult to quantify formally. In this paper we discuss ideas related to using a utility function driven approach to certification of a dynamic resource control system for the DARPA Adaptive and Reflective Middleware Systems (ARMS) program.

---

\* Approved for Public Release, Distribution Unlimited.

The ARMS program is aimed at developing a new generation of capabilities for the dynamic allocation of real-time applications for a concept system which needs to accomplish multiple, possibly conflicting missions. A key attribute of the concept system is the necessity of system-wide distributed resource management strategies embedded and enforced through the system infrastructure. The multiple, concurrent-mission oriented resource management system must run distributed real-time software using resources in a dynamic computing environment with common resources such as processors, bandwidth and memory.

In the distributed ARMS environment software components need to share access to system resources, but there must be some level of assurance that 1) competing QoS concerns will be balanced in order to maintain a sufficient level of overall system quality and 2) the system will best be able to meet its specified, possibly dynamic, operating requirements. For example, even after partial system failures (either hardware or software failures) have decreased the ability of the system to accomplish all aspects of its missions, the system may still be required to accomplish as many key tasks associated with its missions as possible.

Therefore, for situations when partial system failures may restrict resource access, we are developing a Multi-Layered Resource Management (MLRM) system for ARMS to manage resource usage at multiple levels of abstraction. The MLRM allows for the dynamic adjustment of the allocation of resources provided to computational tasks in ARMS in contrast with more traditional static resource allocation strategies that are currently being used. We have designed the MLRM to be a hierarchical resource control system that uses the system's measured application utility as a feedback control signal to dynamically adjust the system's allocation of resources at multiple levels of abstraction.

We define a utility function, called the *Application Utility* function, which is focused on user-perceived elements of derived external value to assess the ability of the MLRM to effectively allocate resources. The utility function is computed in real-time as the system performs its diverse computation jobs. Every computation job that is completed successfully causes an increase in the system's utility, while every failure to complete a computation job causes a decrease in the system's utility. Hence, when partial system failures or a change in operating mode occurs, in order to avoid decreases in the computed utility, the MLRM should make adjust to the resource allocation.

Naturally, information about system failures is not always available to the MLRM, but decreases in the calculated utility can be an indication that failures have occurred which cause the current resource allocation to be insufficient and the system may need to redeploy its resources to accommodate possible failures. By redeploying resources, the system utility would increase if the new allocation allows the system to better accomplish its computation tasks. Consequently, we are using the utility measure as an evidentiary artifact to evaluate how well the MLRM system dynamically adjusts system resources. Previous testing and formal analysis approaches to certification have similar goals as our utility function approach to certification, i.e., to provide quantitative *evidence* of correct system behavior, whether it is a

measure of code coverage (for testing), mathematical proofs (for formal analysis), or collected measures of utility (our assessment).

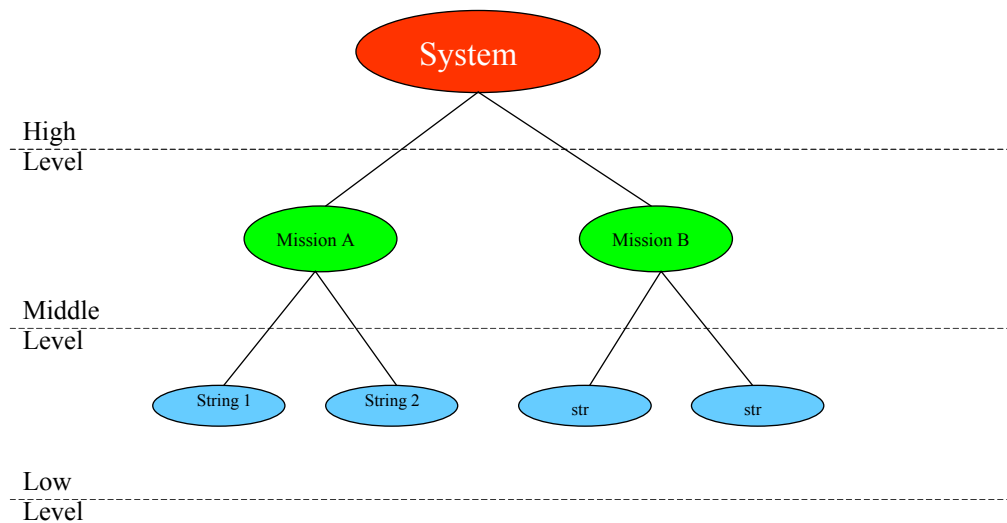
The next section of this paper describes aspects of the ARMS system that are relevant to our efforts at developing a conceptually certifiable MLRM. Section 3 describes utility functions that are used as measures of the quality of resource usage in the system. Section 4 outlines in general terms our control system design that uses the quality measure as a feedback signal in order to dynamically share resources in the system. Section 5 describes some preliminary approaches we have taken to using the utility function as evidence for the certification of the feedback control system in MLRM.

## 2. The ARMS System

One goal of the ARMS project is to design a runtime computational resource allocation engine called MLRM that allows sufficient system resources such as CPUs, memory, and bandwidth to be allocated to accomplish user specified tasks. The ARMS system might have multiple concurrent missions. As specified by the user, some of these missions might be relatively more important than others during different operating modes.

Every mission in the system can generally be decomposed into possibly repeated sub-missions called application strings. For example, a real-time video processing mission might have multiple strings that correspond to the tracking of various objects in the video image. The decomposition of global system behavior into missions and missions into strings is demonstrated visually in Figure 1.

Separate application utility functions are designed for each system layer. The higher layer utility



**Figure 1: System-Mission-String Hierarchy**

functions are composed from lower layer utility functions. It is our contention that this hierarchical utility function approach to integrated dynamic resource management will also provide a more rigorous

underpinning for and potentially easily understood approach to system certification applicable to other distributed, real-time embedded (DRE) systems.

Before introducing the application utility functions and their decomposition in more detail we first formally define some terms:

- Application – An application is an instance of code running on a node.
- Job – An individual unit of information to be processed by possibly multiple applications subject to warfighting QoS requirements.
- Application String – An application string is the path of applications that process a job. A job belongs to exactly one string.
- Mission – A high-level, possibly repeated, objective that is composed of strings.

### 3. UTILITY FUNCTIONS IN MLRM

Now that the ARMS system has been introduced, we present the application utility functions that are used to evaluate the user-perceived value delivered by the system and as evidence for certification purposes. Due to the system-mission-string decomposition, the application utility in a system is computed by taking the weighted sum of the application utilities of the system's missions, and a mission's application utility is computed by summing the application utility of the mission's strings. We first show a method to compute the application utility of a string. Factors contributing to this value include:

- 1) Timeliness – Are the end-to-end deadlines of jobs processed by the strings met? Missed deadlines can have negative, sometimes catastrophic, impact on the system.
- 2) Quality – How accurate or complete is the information or data delivered by an application string? Data compression, filtering, data transformation and incomplete transmission of information can all affect the quality of information delivered by the application string in a potentially adverse manner.

We first investigate how the above factors contribute to the application utility of a string. The factors are then incorporated into a comprehensive utility function for the evaluation of application utility.

#### 3.1 Timeliness

Jobs processed by application strings typically have real-time deadlines associated with them that are either met or missed for each job in a task. Missed deadlines could potentially result in failure of the mission and may preclude certification if sufficiently catastrophic. Therefore, if the deadline is met, then some utility reward of value  $y$  ( $0 \leq y \leq 1$ ) can be assigned for that job. If a hard real-time deadline is missed, there can be a utility penalty  $p$  ( $-1 \leq p \leq 0$ ) associated with missing the deadline.

Suppose  $ec$  is the completion time of a job with an absolute deadline of  $d$ . If  $ec \leq d$  then the job has been completed before its deadline. Conversely, if  $ec > d$ , then a job has missed its deadline. Therefore, for a job with completion time  $ec$  and an absolute deadline of  $d$ , the following functions can then capture the value  $\gamma$  (equation 1a) or the penalty  $p$  (equation 1b) under the scenarios discussed above:

$$\gamma = f_{\gamma}(ec, d) = \begin{cases} y & ec \leq d \\ 0 & otherwise \end{cases} \quad (1a)$$

$$\rho = f_{\rho}(ec, d) = \begin{cases} p & ec > d \\ 0 & otherwise \end{cases} \quad (1b)$$

If the strings in a system miss too many deadlines, utility would drop and provide evidence detrimental to the certification of the system. Note that the penalties and reward associated with jobs may vary from string to string and from mission to mission.

### 3.2 Quality

In addition to meeting real-time deadlines, the quality of the computational processing of a job is also of importance. When data is moved along an application string from producer to end consumer and processed by a number of applications along the way, part of the job's data may be lost, or the data reaching the consumer may become transformed in order to speed processing. For example, data compression and filtering techniques used to reduce bandwidth usage could irrecoverably degrade the quality of the data transmitted between the data producer and data consumer. If the strings in a system are meeting deadlines by processing jobs with a low quality, then the system should be unable to attain a high enough utility that would warrant certification.

Let  $q$  ( $0 \leq q \leq 1$ ) be a measure of the relative change in quality of a set of data after it has been processed by a job. If  $q=1$ , then the data processing performed has caused no data loss. If  $q=0$ , then there has been a complete data loss caused by application processing. The value  $q$  can be measured, reasonably estimated or experimentally determined. There may be multiple methods for measuring a change in data quality. One approach would be to include an information-theoretic entropy measure of data corruption.

### 3.3 Application Utility Functions

We now discuss how to incorporate the above quality and timeliness measures into an integrated application utility function for strings. First, quality functions are designed for individual jobs and strings that take into account timeliness and quality.

If a job is penalized because it missed its deadline, then the quality factor  $q$  is no longer important and the penalty  $\rho$  should be the utility (albeit negative utility) of the job. However, if a job is rewarded with a value  $\gamma$ , then we define the product of  $\gamma$  and  $q$  to be the utility of the job (1 is considered to be the maximum utility of a job). Therefore, the following function can be used to capture the utility  $u_l^{job}$  of the  $l^{th}$  job in a string with respect to timeliness (value or penalty), and quality:

$$u_l^{job} = \rho_l + \gamma_l q_l$$

Recall from the discussion of the timeliness factor in Section 3.1 that either  $\gamma_l$  or  $\rho_l$  will be zero, but not both. Therefore, if  $\rho_l$  is zero, then  $u_l^{job} = \gamma_l q_l$ , and if  $\gamma_l$  is zero, then  $u_l^{job} = \rho_l$ .

In a time frame of length  $T$ , two strings could execute a different number of jobs. Assuming all the jobs in both tasks met their deadlines and had perfect information quality, then one task would have a much higher utility simply because it has a higher throughput. In the context of ARMS, it is desired to measure the utility of a string over a time  $T$  by averaging the utility of all jobs processed by the string during the time interval. If  $P_j$  is the total number of jobs for a string  $j$  in the time frame  $T$  for evaluation, one could compute the average utility  $UA_j^s$  for the string as:

$$UA_j^s = \frac{1}{P_j} \sum_{l=1}^{P_j} u_l^{job}$$

We now show how a string's application utility contributes to the application utility of its mission and hence the application utility of the system. Assume there is a set of  $N_i$  application strings associated with the mission  $i$ . Each string  $j$  has an associated value  $w_j^s$  that indicates the relative user-perceived importance of the application string to the mission. For example, a string with an importance value of 0.5 is considered five times more important to the user than one with an importance of 0.1. Also, the completion of jobs in a string with a relative importance of 0.5 is five times as important towards the certification of the system than the string with an importance value of 0.1.

Therefore, the relative mission level application utility  $UA_i^m$  for mission  $i$  can be defined as the sum of the Application Utilities of mission  $i$ 's strings weighted by their importance values as shown in Equation 6:

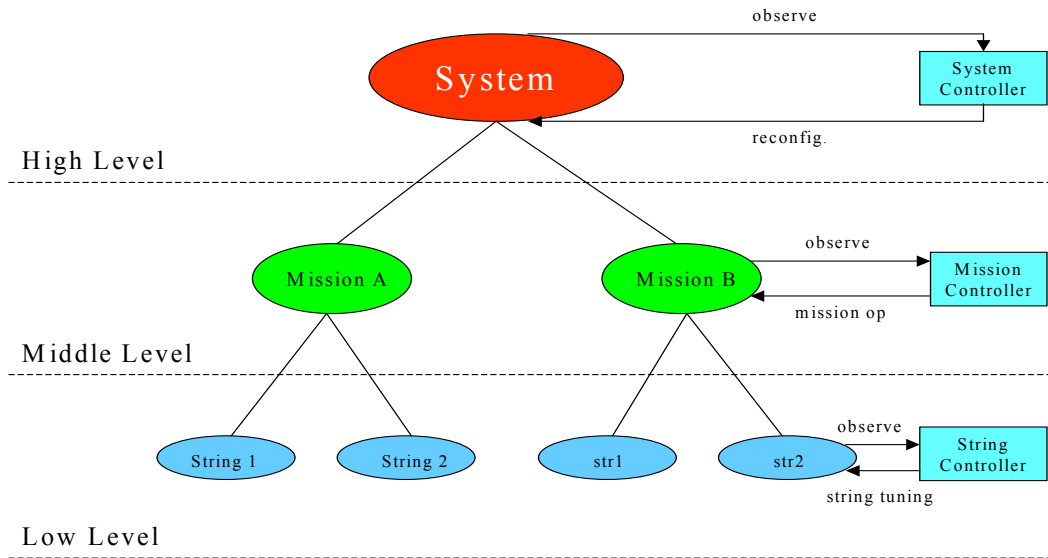
$$UA_i^m = \sum_{j=1}^{N_i} w_j^s UA_j^s$$

If there are  $M$  missions, then the global application utility  $UA^g$  is similarly defined as the weighted sum of the missions' application utilities where the weight  $w_i^m$  is a measure of the relative importance of the mission  $i$ :

$$UA^g = \sum_{i=1}^M w_i^m UA_i^m$$

#### 4. FEEDBACK RESOURCE CONTROL

As described above, in the ARMS MLRM, computational resources are allocated to deployed application strings so that the strings could process their jobs with maximum utility. Some resources, once allocated to strings, may become inaccessible due to resource contention and unpreventable system behavior such as hardware failures. Additionally, the system user may send commands to the controller causing operating mode changes. Some strings that are initially low priority and low resource access may be given a higher priority after a mode change that would warrant them being given more resources. When these situations arise, the resource control system should be able to automatically and dynamically manage resources to maximize the system's utility. This section describes in broad strokes a hierarchical control



**Figure 2: Three-tiered control system hierarchy**

system to dynamically manage resources in the ARMS MLRM based on a system-mission-string decomposition. However, more information about the MLRM control system is available<sup>1</sup>. A partial schematic of the system-mission-string hierarchy of the control system's operation can be seen in Figure 2.

At the highest level, the system controller has the ability to trigger system-wide full reallocations of resources affecting multiple missions. There is likely to be a relatively high cost associated with performing such a system wide action, so global allocations should only be performed when the cost of these actions can be justified by a sufficiently large gain in application utility or when no other effective actions are available. These system level reallocations may be driven by command decisions, or triggered as an automatic response to changes in the system that could not be mitigated by lower level controls.

At the mission level of operation, the mission controller manages the behavior of strings to maximize the local mission-level application utility. Possible actions that could be taken at the mission level include moving the location of string operations between pools after failures or in response to a change in operating mode.

The application utility functions defined in Section 3 are computed at the lowest level via the summation of string level utilities. We therefore start from the proposition that string level control actions are naturally used at the lowest levels of the control hierarchy. Strings can locally and independently tune their operations in order to locally maximize their application utilities by adjusting their controllable parameters such as processing quality. Generally there is little or no cost associated with this low-level tunings, so the strings are allowed to perform these actions freely bounded only by their resources limits and desired operational ranges.

At all levels of the MLRM hierarchy (system-mission-string), application utility is used as a feedback control signal, which should be maximized over the course of system behavior and in the face of adverse conditions such as partial system failures.

## 5. Utility Measurements and Evidence-Based Certifiability

There is a discernable relation between the assessment utility measures we are developing, their use in control of DRE systems, and certification processes for these systems. As discussed in the introduction, traditional testing and formal analysis methods face difficulties when being applied to dynamic systems because it is more difficult to identify a finite set of tests that would cover system operation. We believe that utility measures and utility-driven control functions can be used as a tool for certification of dynamic systems. While these explorations are ongoing, the following is a list of the ideas, directions, and issues we have also identified:

- Utility functions can capture all the attributes (or a large set of them) of higher or lower utility, without needing to capture all the factors contributing to them. For example, a utility measure can recognize that utility is negatively affected by missing deadlines, without needing to identify what can cause missed deadlines (such as network outages, denial of service attacks, improper scheduling, resource overload, hardware or software failures, and so on). This provides a quantitative measure for certification that is achievable even in highly open and dynamic environments.
- Feedback controllers driven by system utility lend themselves to Monte Carlo simulations to gain statistical evidence of the system's ability to maintain correct behavior under different conditions.
- Application utility is a measure of the user-perceived value derived from using the MLRM and gives evidence for how well the MLRM can respond to changes in system operating modes. We can combine that with mission-derived limits on system operating modes, which focuses the certification effort on verifying the controller's ability to enforce correct behavior within these limits. A controller based design provides convenient and easy to understand places within the architecture to implement simplified control "limiters" which may be amenable to be certified by inspection or formal methods.
- Evidence of properly restricted feedback interactions between a controller and a system ease the difficulty of certifying the overall controlled system.

---

<sup>1</sup> K. Rohloff, J. Ye, J. Loyall and R. Schantz. A Hierarchical Control System for Dynamic Resource Management. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, San Jose, CA, April 2006.