A Framework for Using Benefit Functions in Complex Real Time Systems

David Andrews
Ravi Vemuri
Electrical Eng. & Computer Science
ITTC
University of Kansas
Lawrence, KS 66045
mailto:dandrews | vemuri@ittc.ukans.edu

David M. Chelberg
David Fleeman
David Parrott
Lonnie R. Welch
School of EECS
Ohio University
{ chelberg | david.fleeman |
david.parrott | welch } @ohio.edu

Scott Brandt
Computer Science Department
Jack Baskin School of Engineering
University of California, Santa Cruz
sbrandt@cs.ucsc.edu

Abstract

Researchers are currently investigating applying benefit, or utility functions for allocating resources in limited, soft real time systems [1,2,3]. While the future of real -time computing research and practice will likely exploit the utility of benefit-based models, this will not occur until a suitable system level benefit framework has been defined. This paper provides an overview of ongoing work to develop such an initial framework that can formalize the use of benefit functions in complex real time systems. It is easily shown that this framework can support traditional hard/firm/soft real -time paradigms as well as support newer proposed models for future real time operation [5,6]. We also show that our proposed framework unifies composition of benefit values derived from heterogeneous perspectives, including those derived from the application perspective.

1: Introduction

Many researchers have been investigating mechanisms for better supporting soft real time processing [1,17, 18,19,20,21,22] where some or all of the applications can miss deadlines and still perform satisfactorily. This type of situation can occur when the worst -case resource requirements of all running applications cannot be simultaneously met. This scenario has become more common as embedded systems designers have distributed previously stand -alone systems. This networking can introduce asynchrony and non -stochastic data loading and subsequent processing requirements. In these new systems, it is the respons ibility of the applications to run correctly with less than optimal allocated resources. Additionally, the resource allocation may be time varying based on changing mission scenarios or faults.

New operational modes are being employed that allow applicati ons to specify their relative importance within the overall system in terms of utility or benefit. This benefit can then be used by the system to guide the dynamic allocation or assignment of resources that seeks to optimize the global system benefit. Di fferent approaches are being investigated to represent application benefit to resource allocation managers. It is the applications' responsibility to decide the best usage of limited resources. Some applications may choose not to alter their prescribed p rocessing, which can lead to missed deadlines. Other applications may choose to alter their processing in order to return degraded results without missing deadlines. Hybrid solutions are also possible. In each of these cases, the application requires a mechanism to establish its degraded but achieved benefit or utility to the overall system.

Benefit is defined in [1] as varying functions representing the value to a system for completing a process at a certain time. In [1], benefit functions were used a s alternative scheduling algorithms for optimization in systems running soft real time processes. Figure 1 shows generic time varying benefit curves. The first benefit curve shows the benefit of performing some task is constant in the specified time int erval between t_{start} and t_{end}, and executing the task outside this window does not return any benefit. The second curve shows a constant benefit between t_{start} and t_1, and a diminishing benefit between t_2 and t_3.

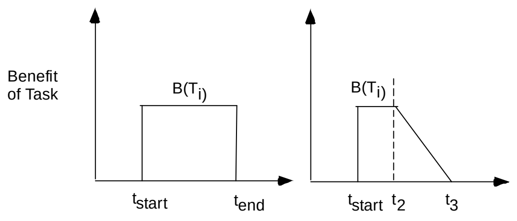

Figure 1: Benefit Functions

Researchers [16] hav e continued to extend Jensen's original definition of time varying benefit functions into the realm of dynamic real -time scheduling decisions for soft real time processes. The base notion of a benefit function has been extended to control the dynamic all ocation of resources based on overall achieved system benefit. In this interpretation, benefit relates the quality of the output of an application achieved by running with a given resource allocation. The differing quality of output is a result of running different algorithms at each resource allocation, or a difference in the resources allocated and not a result of missing deadlines by greater or lesser amounts. A missed deadline is used to indicate that an application is not satisfactorily executing its algorithm within allocated resources and triggers changes in algorithm or resource allocation to correct the problem.

At the lowest level of abstraction, benefit can be viewed as a function of time, where applications have varying benefit based on the ti me instant at which they complete their processing relative to their deadlines. If $b_i(r)$ is the benefit function for application i as a function of the amount of allocated resources and $r_i(t)$ is the actual resource allocation provided to the application I at time t, then we can define $B_i(t) = b_i(r_i(t))$ to be the actual benefit provided by application i at time t, or the *output benefit* of application i. Given the output benefit of applications over a span of time, we define our specific benefit metrics as follows.

At any particular time t, $B_i(t)$ is the *instantaneous benefit* of application i. During any interval of time (t_1, t_2), the *total benefit* is defined as

$$B_i(t_1,t_2) = \int B_i(t)\,dt.$$

With functions on discrete values, the benefit curve from t_1 to t_2 is a step function with m distinct intervals starting at times t_j and the integral is a discrete sum,

$$B_i(t_1,t_2) = \Sigma\, B_i(t_j) \times (t_{j+1} - t_j)$$

The average benefit $\mathit{B}_i(t_1,t_2)$ from time t_1 to t_2 is simply $(1/t_2-t_1)B_i(t_1,t_2)$. The instantaneous, total, and average benefit for a set of running applications $B(t)$, $B(t_1,t_2)$, and $\mathit{B}(t_1,t_2)$ are simply the sum of the individual application benefits.

Benefit gives a good, but incomplete, indication of the quality of the output of a process. In particular, moving from one output benefit to another may indirectly affect the quality of an application. As an example, a video player that continuously runs at 15 frames per second may be preferable to one that alternates between 0 and 30 frames per second every few seconds, even though their average benefit will be the same. To measure this variation between levels we define the *instability* of each application as follows.

At any time t, the *instantaneous instability* of application i is the slope of the curve of the output benefit, thus $I_i(t) = |B'_i(t)|$. For applications defined on discrete resource amounts, the slope is infinite during the transition from one level to another and 0 at all other times. We therefore define the instantaneous instability to be $|B_i(t)-B_i(t+\varepsilon)|$ for some suitably small ε. During the interval of time (t_1,t_2), the *total instability* is defined as $I_i(t_1,t_2) = \int I_i(t)\,dt$. As above, for applications defined on discrete resource amounts the benefit curve from t_1 to t_2 is a step function with m distinct intervals each starting at time t_j, and the integral is a discrete sum, $I_i(t_1,t_2) = \Sigma I_i(t_j)$. Conceptually, the instability of an application from time t_1 to t_2 is simply the amount of benefit change during that time. The average instability $I(t_1,t_2)$ of application i from time t_1 to t_2 is $(1/t_2-t_1)I_i(t_1,t_2)$. Instantaneous, total, and average instability for a set of running applications ($I(t)$, $I(t_1,t_2)$ and $I(t_1,t_2)$) are defined as the sum of the values for the individual applications.

Although benefit models have been shown to be effective [4][7][9] for the integration of small numbers of processes, they have not been applied to the integration of large numbers of processes in complex relationships. Previous investigations of benefit functions have focused on developing static benefit functions that can be used to direct scheduling of application processes based on projected resource availability. These static functions can be used to relate potential benefit based on projected resources. By its nature, this approach is based on *a priori* knowledge of available resources. This assumption has been adequate for legacy embedded systems, where worst case execution times are known, and schedules can be developed to support the worst case execution times. Modern systems are now incorporating more networked capabilities, allowing data sharing and data fusion between subsystems. While this promises increased capabilities in the form of better knowledge formation, it also has introduced more non-determinism in resource availability. This non-determinism challenges the basic assumption of *a priori* knowledge of resource availability. Dynamic monitoring techniques are being investigated to determine the instantaneous resource availability in order to direct scheduling.

The abstract notion of benefit can certainly extend to these new complex systems with many layers and many different types of benefit [10][8][13]. These include application notions of benefit, single system notions of benefit involving resource tradeoffs within that system, and aggregate system notions of benefit involving tradeoffs across systems and based on complex relationships between the systems and between the applications running on those systems. In such complex systems, one cannot simply maximize the benefit of each individual application and system and hope to achieve optimal system performance.

Without loss of generality we develop a framework for representing benefit that interprets the traditional concepts of "hard", "soft", and "firm" within an algebraic framework supporting composition of different levels of benefit abstraction. Our framework encourages the premise that at any instant of time benefit within a complex enterprise system is a function of many interrelated entities that vary according to complex rules and interactions among different subsystems.

The different levels of abstraction at which a calculated benefit can affect aggregate system behavior is easily understood. At the lowest level of abstraction benefit can be viewed as a function of time, where applications have varying benefit based on the time instant at which they complete their processing relative to their deadlines. However, benefit may also be viewed as a function of the completion times and the resources allocated to the processes, as a function of resources allocated and of the quality of the output produced at that resource level, or solely based on the quality, frequency, or other properties of the output.

The perspective from which benefit is calculated is also important. Benefit may be calculated from the application perspective. It may also be calculated from the single system perspective, from which summed benefit of the set of applications is the important measure, but spare resources may also be important. In this case, the spare resources themselves can be characterized with a benefit function and included in the calculations [8]. From a complex system perspective, however, there are many tradeoffs to be made in terms of CPU usage, network bandwidth, individual system capabilities, etc., and simply summing the benefit of the individual systems will not capture all that matters in such a complex system.[13] Finally, there is also a notion of end-user benefit in many systems, and what the user actually cares about may differ markedly from what has been specified by the developer.

Because the abstract notions of benefit differ in important respects and because many different notions can be expected in complex real-time systems of the future, we believe that it will not be possible to maximize benefit through a simple optimization as is done in current soft real-time systems. Accordingly, we are developing a framework for describing, composing, and combining many notions of benefit with the goal of being able to optimize benefit in highly complex real-time systems.

2: Formalizing Benefit

We define a value of benefit associated with attempting to meet some specified objective as a point in the interval:

$$b_i = [l_i, 0, u_i] \text{ such that } l_i \leq 0 \text{ and } 0 \leq u_i$$

For observed objective i, $b_i = 0$ denotes a quiescent state associated with the objective exactly meeting a specified requirement. If the benefit value $b_i > 0$, this denotes that the objective is not only meeting its requirement, but is also providing additional capability beyond its minimum. If the value $b_i < 0$, this denotes the objective has dropped below its minimum requirement, but is still returning a diminished benefit value. For a traditional “hard” real time objective, a limitation on the acceptable value of benefit can be interpreted as interval $[0, .. +\infty]$. For “soft” real time objectives, the benefit value b_i can assume any value within the interval $[-\infty, .. 0 .. +\infty]$. The lower and upper limits, l_i, u_i respectively, quantify limits for benefit optimization calculations. l_i represents the lowest allowable value of benefit that the objective is allowed to return. u_i represents a point at which the value of benefit returned is at the upper limit and any additional benefit is superfluous or unachievable. Optimizing benefit is viewed as conditionally maximizing the value of the benefit vector subject to each benefit value being in $l_i \leq b_i \leq u_i$.

As a simple example, we can optimize benefit by summing the individual values as:

$$\text{Max } B = \text{Max } \{ \Sigma b_i \mid l_i \leq b_i \leq u_i \}$$

Where our search space is bounded by

$$\Sigma l_i \leq \Sigma b_i \leq \Sigma u_i.$$

More complex benefit calculation functions based on heuristics appropriate for a specific system can also be used. Using the definitions above, we can compose a benefit vector $B = <b_0, b_1, b_2, ... b_n>$ with defined associated limit vectors

$$L = <l_0, l_1, ... l_n> \text{ and } U = <u_0, u_1, ... u_n>.$$

It should be noted that algorithms used to optimize benefit should be relatively insensitive to small changes in benefit values. Otherwise, small fluctuations in benefit may cause large changes in the manner of pursuing system objectives.[23] To accomplish this, we can include a term in the benefit vector representing the value of doing nothing.

Our definitions support hierarchical composition of benefit vectors, and allow abstract composition and optimizations to be performed.

As discussed above, a simple way to compute maximum benefit in hierarchical systems is to sum the individual vectors. For example, we can sum each objective, i, in each subsystem, j:

$$\text{Max } B = \text{Max } \{ \Sigma \Sigma b_{ij} \mid l_i \leq b_i \leq u_i \}$$

To ensure that we give more resources to specific individual objectives, we can associate a weighting factor (importance, otherwise known as significance in [14] and priority in [15]) for each objective in the summation:

$$\text{Max } B = \text{Max } \{\Sigma\, \Sigma\, w_{ij} b_{ij} \mid l_i \le b_i \le u_i \}$$

Another simple way to compute maximum benefit in a hierarchical system that ensures at least some amount of performance in each objective is to maximize the minimum benefit:

$$\text{Max } B = \text{Max } \{ \text{Min } \{ \text{Min}\{ b_{ij} \mid l_i \le b_i \le u_i \} \} \}$$

It is worth noting that benefit functions can be used for analysis during the initial system design by monitoring the system's ability to meet each specified lower level of benefit.

Also, slack can be added to the benefit vector to assure that every resou rce is used at the minimal amount. This allows the system to handle a sharp increase in resource utilization in a graceful manner.

2.1: System Definitions Using Benefit

We provide the following definitions for our benefit framework.

Definition 1: A "Hard" real time system is defined as

$$H_{sys} = \{ \forall\, i \mid l_i = 0\}$$

Simply stated, a hard real time system is a system such that the minimum acceptable levels of benefit are not allowed to be negative.

Definition 2: A system is quiescent if

$$\text{Quiescent}_{sys} = \{\forall\, i \mid b_i = 0\}$$

A system is in a quiescent, or steady state, when all measurable benefit is exactly returning the expected benefit value.

Definition 3: A system's performance is formally correct if

$$\forall i \mid b_i \ge l_i \text{ for all time t.}$$

This definition relates the system's ability to always meet minimal acceptable levels of performance.

Definition 4: A system is under-constrained if

$$\Sigma b_i \ge 0 \text{ and } \forall i \;\; b_i \ge l_i.$$

This denotes that the system is meeting each individual benefit minimum, and the aggregate is providing additional benefit beyond the quiescent capabilities for which it was designed. This is characteristic of a system with exce ss capacity.

Definition 5: A system is over-constrained if

$$\Sigma b_i \le 0 \text{ and } \forall i \;\; b_i \ge l_i$$

Note that a system may be over-constrained, but still adhere to a formally correct performance behavior.

Lemma 1: A system with all benefit values classified as "hard" c annot be over -constrained and still formally meet its performance requirements.

3: Application Examples

The utility of benefit functions allows their use within a system to reflect the specific operational paradigm of the system without loss of generality. C onsider the following example of a function that acts as a “bridge” node within a distributed system. This function is a periodic task that attempts to transmit as many messages as possible between two messaging channels within a given time limit. Once t he task has reached the time limit, a benefit value is calculated based on the number of messages it successfully transmitted. This capability is obviously dependent on the availability and performance of the interconnection networks, and may vary with time.

Function Bridge

```
Benefit = 0;
while(true){
cnt = 0;
while(current_time < limit){
   buffer = receive(ch1)
   send(ch2,buffer)
   cnt += 1;
}
if (cnt == max) IPC.benefit = 0;
elseif(cnt < max) IPC.benefit = -Δ;
else IPC.benefit = +Δ;
next_time = limit + interval;
suspend(next_time);
}
```

In this example, each process defines a benefit Δ, representing the relative benefit to the system for over or under performing. Note that in this example, the function is not a llowed to change the lower limit of acceptable performance. Instead, the lower limit is set by the system at initialization, and is maintained by the separate IPC benefit monitor as shown below. In this mode, the tasks are passive and only report achieve d benefit. Resources are allocated by the IPC monitor based on measured benefit from all functions performing messaging operations. The specific allocation scheme is dependent on the system.

_IPC_Performance_Monitor

```
If(benefit[task] < l[task]){
   if(excess > Δ){
      resource[task] += Δ;
      excess -= Δ;
   }
   else
      task_lowest = f(greatest_excess);
      resource[task_lowest] -= Δ;
      resource[task] += Δ;
   }
elseif(benefit[task] > l[task])
   resource[task] -= Δ;
   excess += Δ;
}
else(benefit[task] == l[task]);
```

In this simple example, the IPC Performance Monitor maintains a variable that represents the current excess capacity of the resources, and optimizes the excess capacity. If a task reports additional benefit, the monitor will decrease the resource and increase excess capacity. This simple algorithm optimizes the systems ability to meet minimal requirements and respond to dynamic fluctuations in requests.

This example shows a simple linear combination of benefits. The advantage of using this approach allows the application of more complex and system specific methodologies for combining and calculating individual and system benefits.

3.1: Image Processing System

As stated in the introduction, benefit may be viewed as a function of resources allocated and of the qualit y of the output produced at that resource level. In the following example we apply this view of benefit from an application perspective, and then combine the benefit derived from each application to view benefit from a system perspective.

The Image Proces sing System (IPS) [11,12] is a simulated satellite system that has four applications: a Camera Simulator (CS), an Image Processing Agent (IPA), a Compressor Agent (CA), and a Download Agent (DA). The Camera Simulator produces images of the earth at some specified resolution and image rate. These images are processed by the Image Processing Agent to determine the cloud cover percentage of the image. If the cloud cover percentage is high, the image yields little scientific benefit, and it would be prefera ble for the Compressor Agent to either discard the image or use lossy compression before storing the image in the onboard storage buffer. The Download Agent is simply responsible for downloading the images with highest scientific benefit from the satellit e to the ground when network downlink time is available. We will ignore the Download Agent application in the rest of this paper for the purpose of simplicity.

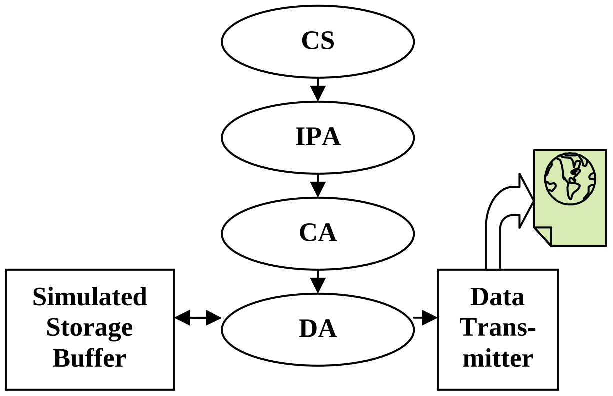

Figure 2: Image Processing System Prototype

We now introduce the term *service level parameter*. A service level parameter is a parameter to an application or system that can have an impact on both how many resources are used and how much scientific benefit is achieved for the user. There are two types of service level paramete rs: mutable and immutable. A mutable parameter is one such that an external resource manager can adjust its value to achieve desired results. An immutable parameter is usually fixed by an application at run-time or is a function of data -dependent input. If enough resources are present in the system, the mutable service level parameters would always be set to their best level. The presence of mutable service level parameters allows a resource manager to cause an application to use less resources in the c ase of an overload or to meet real-time constraints.

In our example, the resolution and the rate that the Camera Simulator sends images can be set by a resource manager. The pixel percentage service level parameter can also be set to change how many pixel s the Image Processing Agent will process when determining cloud cover. The compression percentage can be set to change the type of compression to be performed by the Compressor Agent. All of these changes can affect the resource usage and the derived us er benefit. Also note that the cloud cover percentage of the image cannot be controlled by the resource manager, but does affect the scientific benefit of an image and the compression algorithm used.

```
ips.sl_parameters={image_interval,resolution,cloudcover_%,pixel_%,compression_%  }
```

We define each service level parameter as a tuple that defines 1) whether it is mutable or immutable, 2) whether it is continuous or discrete, and 3) the values that the parameter can assume. The image_interval parameter is mea sured in seconds and represents how often a new image should be taken.

```
ips.sl_parameters(image_interval) = < MUTABLE, CONTINUOUS, [30, 60] >

ips.sl_parameters(resolution) = < MUTABLE, DISCRETE, { LOW, HIGH } >

ips.sl_parameters(pixel_%) = < MUTABLE, CONTINUOUS, [0, 100] >

ips.sl_parameters(cloudcover_%) = < IMMUTABLE, CONTINUOUS, [0, 100] >

ips.sl_parameters(compression_%) = < MUTABLE, CONTINUOUS, [0, 100] >
```

We will now tie this example system to our proposed benefit framework. For this example, we are assuming that each service level parameter independently affects benefit. This assumption is for simplicity, and we are researching other alternatives for combining service level parameters. Another simplifying assumption we make is that the benefit of each service level parameter can be linearly mapped to a benefit range.

In order to know what combinations of service level parameters yield the most benefit, a user must specify the benefits for each service level parameter. As this framework proposes, the scientist defines that each service level parameter maps onto a range $[l_i, 0, u_i]$ such that $l_i \leq 0 \leq u_i$, and defines the quiescent value in that range. For continuous, immutable service level parameters, the quiescent value = $l_i = 0$, and $u_i = 1$. The parameter values will map linearly onto this interval. For continuous, mutable service level parameters, $l_i = -1$ and $u_i = 1$. The parameter values will be divided into two sections, the group below quiescent, and the group above quiescent. These values will map linearly onto their respective benefit intervals. For discrete service level parameters, the complete mapping from value to benefit must be specified.

ips.sl_parameters(image_interval).value = 60 → $l_i = -1$
ips.sl_parameters(image_interval).value = 45 → $q_i = 0$
ips.sl_parameters(image_interval).value = 30 → $u_i = 1$

ips.sl_parameters(resolution).value = LOW → $l_i = q_i = 0$
ips.sl_parameters(resolution).value = HIGH → $u_i = 0.1$

ips.sl_parameters(pixel_%).value = 0 → $l_i = -1$
ips.sl_parameters(pixel_%).value = 50 → $q_i = 0$
ips.sl_parameters(pixel_%).value = 100 → $u_i = 1$

ips.sl_parameters(cloudcover_%).value=100 → $l_i = q_i = 0$
ips.sl_parameters(cloudcover_%).value = 0 → $u_i = 1$

ips.sl_parameters(compression_%).value = 100 → $l_i = -1$
ips.sl_parameters(compression _%).value = 75 → $q_i = 0$
ips.sl_parameters(compression _%).value = 0 → $u_i = 1$

The second step is defining what parameters affect each application. The Camera Simulator is affected by resolution and image_interval. The Image Processing Agent is affected by resolution and pixel_percentage. The Compressor Agent is affected by resolution, cloudcover_%, and compression_%.

Now that we have defined the affect of service level parameters on the benefit to the user, we will apply the two hierarchical methods introduced in section 2 to derive both individual application and system benefit values, and compare the results of using each method. To do a comparison we must choose some setting of the service level parameters.

```
ips.sl_parameters(image_interval).value = 45
ips.sl_parameters(resolution).value = LOW
ips.sl_parameters(pixel_%).value = 75
ips.sl_parameters(cloudcover_%).value = 40
ips.sl_parameters(compression_%).value = 60
```

Applying the first hierarchical technique (average), we have the following application and system benefits if we assume that all $w_i = 1$:

```
ips.cs.benefit = (0 + 0)/2 = 0
ips.ipa.benefit = (0 + 0.5)/2 = 0.25
ips.ca.benefit = (0 + 0.6 - 0.2)/3 = 0.13
ips.benefit = (0 + 0.25 + 0.13)/3 = 0.13
```

Now applying the second hierarchal technique (min), we have the following benefits:

```
ips.cs.benefit = min(0, 0) = 0
ips.ipa.benefit = min(0, 0.5) = 0
ips.ca.benefit = min(0, 0.6, -0.2) = -0.2

ips.benefit = min(0, 0, -0.2) = -0.2
```

As you can observe, the first approach indicates the collective values of the service level parameters produces benefit that is above the quiescent state of the system. However, the second approach indicates that the service level parameter settings yield a benefit that is significantly below the quiescent state. Depending on the characteristics of the real-time system being modeled, one of these algorithms may more accurately compute the perceived user benefit. It appears that the first approach may be more useful for systems with soft constraints, and the second approach may be more useful for systems with hard constraints.

Another important observation is that if all the service level parameters are equal to their respective quiescent values, then the application and system benefit are also at their respective quiescent values regardless of the hierarchical technique used. This behavior is a desired property of any hierarchical technique that could be used.

4: Conclusion

While the future of real-time computing research and practice will likely exploit the utility of benefit-based models, there are many issues, which must be resolved. This paper provides an initial framework to formalize the use of benefit functions in complex real time systems. It is easily shown that this framework can support the traditional hard/firm/soft real-time paradigms as well as supporting new models for real time operation. We provided a framework that unifies the various types of benefit, including those from the application perspective.

5: Acknowledgements

This work has been partially funded by NASA grant number NAG5-10349.

6: References

[1] E. D. Jensen, C. D. Locke and H. Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems," *Proceedings of the Real-Time Systems Symposium*, 112-122, IEEE CS Press, 1985.

[2] Scott A. Brandt and Gary J. Nutt, "Flexible Soft Real-Time Processing in Middleware," *Journal of Real-Time Systems,* Kluwer, 22(1,2):77-118, January-March 2002.

[3] L. R. Welch, B. Ravindran, B. Shirazi and C. Bruggeman, "Specification and analysis of dynamic, distributed real-time systems," in *Proceedings of the 19th IEEE Real-Time Systems Symposium,* 72-81, IEEE Computer Society Press, 1998.

[4] Giorgio Buttazzo, Marco Spuri and Fabrizio Sensini, "Value vs. Deadline Scheduling in Overload Conditions," in *Proceedings of the 19th IEEE Real-Time Systems Symposium,* IEEE Computer Society Press, 1998.

[5] Edward Lee, "What's Ahead for Embedded Software?", IEEE Computer, Sept. 2000, pp. 18-26

[6] Jason Hill, Robert Szewczyk, Alex Woo, Seth Hollar, David Culler, Kristofer Pister, "System Architecture Directions for Networked Sensors", Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX) Nov. 2000, pp. 93-104

[7] J. Hansen, J. Lehoczky and R. Rajkumar, "Optimization of Quality of Service in Dynamic Systems", Proceedings of the 9th International Workshop on Parallel and Distributed Real Time Systems, April 2001

[8] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, J. Hansen, "A Scalable Solution of the Multi-Resource QoS Problem", Proceedings of the IEEE Real-Time Systems Symposium, Dec. 1999

[9] T. Abdelzaher, E. Atkins, and K. Shin, "QoS Negotiation in Real-Time Systems and its Application to Automated Flight Control," *Proceedings of the 3 rd IEEE Real-Time Technology and Applications Symposium*, June 1997.

[10] E. D. Jensen, "Asynchronous Decentralized Realtime Computer Systems," *in Real-Time Computing – NATO Advanced Study Institute on Real-Time Computing*, NATO ASI Series, Series F, Computer and Systems Sciences, v.127, pages 347-371, Springer-Verlag, 1994.

[11] Ohio University and NASA GSFC, *Hierarchical Agent-based Real-time Technology (HART)*, 2002, <http://rm.ece.ohiou.edu/~hart/>, (11 Jul. 2002).

[12] Shikha Jain, Lonnie R. Welch, David M. Chelberg, et al, .Collaborative Problem Solving Agent for On-Board Real-time Systems., Proceedings of 16th International Parallel and Distributed Processing Symposium, 15-19 April 2002, Ft. Lauderdale, Florida.

[13] A. Burns, D. Prasad, A. Bondavalli, F. Di. Giandomenico, K. Ramamritham, , J. A. Stankovic, and L. Strigini, The Meaning and Role of Value in Scheduling Flexible Real-time Systems *Journal of Systems Architecture* Vol. 46 (2000), p. 305-325.

[14] N.R. Howes, J.D. Wood, A. Goforth, " ***The Peer Tasking Design Method*** ", Proceedings of the Third Workshop on Parallel and Distributed Systems', IEEE CS Press, April 1995, pp 20-29.

[15] G. Buttazzo, M. Spuri, F. Sensini, "Value vs. Deadline Scheduling in Overload Conditions" , Proceedings of the 15th Real -Time System Symposium (RTSS 95), Pisa, Italy, pp. 90-99, December 1995.

[16] Lei Chen, Shahadat Khan, Kin F. Li, Eric G. Manning , "Building an Adaptive Multimedia System using the Utility Model" Parallel & Distributed Processing, Lecture Notes in Computer Science 1586, Springer Verlag, pp 289-298, ISBN 3-540-65831-9. (Int'l Workshop on Parallel & Distributed Realtime Systems (WPDRTS 7), ACM/IEEE/US Navy Surface Warfare Center), San Juan PR, April 1999.

[17] S. Brandt, G. Nutt, T. Berk, and M. Humprey, "Soft Real -Time Application Execution with Dynamic Quality of S ervice Assurance", Proceedings of the 6th IEEE/IFIP International Workshop on Quality of Service, pp. 154-163, May 1998

[18] C. Fan, "Realizing a Soft Real-Time Framework for Supporting Distributed Multimedia Applications", Proceedings of the 5 th IEEE Workshop on the Future Trends of Distributed Computing Systems, pp. 128-134, Aug. 1995

[19] C. Lee, R. Rajkumar and C. Mercer, "Experience with Processor reservation and Dynamic OsS in Real -Time Mach", Proceedings of Multimedia Japan, March 1996

[20] J. Ni eh and M.Lam, "The Design, Implemenation and Evaluation of SMART: A Scheduler for Multimedia Applications", Proceedings of the 16th ACM Symposium on Operating Systems Principles, Oct. 1997

[bra24] R. Rajkumar, C. Lee, J. Lehoczky and D. Siewiorek, "A Res ource Allocation Model for QoS Management", Proceedings of the IEEE Real-Time Systems Symposium, December 1997

[22] H. Tokuda, and T. Kitayama, "Dynamic QoS Control based on Real -Time Threads", Proceedings of the 3rd International Workshop on Network and Operating Systems Support for Digital Audio and Video, Nov. 1993

[23] E. Huh and L. Welch, "An Efficient Schedulability Analysis Policing Technique for Periodic, 5.Dynamic Real -Time Applications," The 10th Workshop on Parallel and Distributed Real-Time Systems, April 2002