

A Pattern Language for Engineering Dynamic Real-Time Applications*

Toni Marinucci¹, Lonnie R. Welch¹, Michael W. Masters² and Paul V. Werme²

1. Introduction

A dynamic real-time system (DRTS) can be defined as a system whose resource requirements are dependent on conditions in a time-varying external environment. Therefore, the resource needs of a DRTS cannot be characterized accurately at design time (see [7]-[11], [13]).

An example of a DRTS is a cruiser or destroyer ship that uses a sensor to detect tracks within a specified area around the ship. An application onboard the ship needs to determine whether each track is a threat. The amount of resources required to determine whether the tracks are threats depends on the number of aircraft in the environment. Similarly, programs responsible for engaging the threats require an amount of the ship's computing resources that is a function of the amount of threats that are being tracked. The ship will encounter varying amounts of tracks during its lifetime, but no matter what the workload, all traffic must be processed and reactions must occur in a timely manner; in other words, it should provide load invariant real-time performance.

This paper provides a pattern language for developing real-time systems that are able to efficiently and effectively operate in dynamic environments. It describes patterns for creating reusable middleware solutions for QoS and resource management and patterns for engineering dynamic real-time systems that employ the middleware patterns.

The Alexander pattern form [1] is used to describe the patterns. Each pattern description provides the following:

“First, there is a picture, which shows an archetypal example of that pattern. Second, after the picture, each pattern has an introductory paragraph, which sets the context for the pattern, by explaining how it helps to complete certain larger patterns. Then there are three diamonds to mark the beginning of the problem. After the diamonds there is a headline, in bold type. This headline gives the essence of the problem in one or two sentences. After the headline comes the body of the problem. This is the longest section. It describes the empirical background of the pattern, the evidence for its validity, the range of different ways the pattern can be manifested... Then, again in bold type... is the solution—the heart of the pattern—which describes the...relationships which are required to solve the stated problem in the stated context. This solution is always stated in the form of an instruction—so that you know exactly what you need to do, to build the pattern. Then, after the solution, there is a diagram, which shows the solution in the form of a diagram, with labels to indicate its main components.

After the diagram, another three diamonds, to show that the main body of the pattern is finished. And finally, after the diamonds there is a paragraph which ties the pattern to all those smaller patterns in the language, which are needed to complete this pattern, to embellish it, to fill it out...”

¹ Center for Intelligent, Distributed and Dependable Systems, School of EECS, Ohio University, Athens, Ohio 45701, {toni.marinucci|welch}@ohio.edu.

² Naval Surface Warfare Center, Dahlgren, VA 22448, {MastersMW|WermePV}@NSWC.NAVY.MIL.

*Supported in part by funding from DARPA and NASA.

Note that the forces for each problem are contained in the introductory paragraph, headline, and body sections of the pattern descriptions. In the Alexander form, each pattern is marked with one or two asterisks. Two asterisks mean that we believe that we have identified a solution that is common to all DRTSs. One asterisk means that we believe that we have made progress toward identifying a solution, but that this may not be the only way to solve a particular problem. Finally, the Alexandrian form also provides a summary of the pattern language prior to the presentation of the patterns.

2. Summary of the Language

The overall problems examined in this paper focus on how to standardize solutions for meeting QoS requirements in dynamic environments, while separating application functionality from these solutions, in order to ease development, testing, and maintenance of DRTSs.

The pattern language is presented as two sets of patterns. The first set provides reusable middleware solutions that enable DRTSs to provide load invariant real-time QoS via adaptive resource management. The reusable solutions are important, because anyone who engineers a dynamic real-time system will encounter problems similar to the ones covered by the patterns, which are:

- 3.1 Allocation Manager³
- 3.2 Resource and QoS Monitor
- 3.3 Resource Allocation Controller

The second set of patterns details how DRTSs can be engineered so that these reusable solutions can be employed. The patterns deal with making the a priori and a posteriori information about DRTSs known. The set of patterns that pertain to a priori information are:

- 4.1 System Specification
- 4.2 System Profile

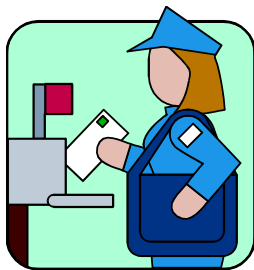
And, the pattern that deals with gathering a posteriori information is:

- 4.3 QoS Information Reporter

3. Reusable Solutions for QoS and Resource Management

This section describes what is needed to build reusable solutions that provide a DRTS with adaptive behavior. In order to achieve real-time performance, the solutions address the problems of determining when and how to adapt.

3.1 Allocation Manager**



The Allocation Manager fulfills the goal of providing a reusable solution for making dynamic real-time systems adaptive in the way they use resources, so that they can effectively deal with dynamic environments.



There needs to be a way to determine if a DRTS is violating any of its QoS constraints; additionally, there needs to be the ability to

³ The numbering corresponds to the subsection in which each pattern is discussed.

determine the root of the problem and make a plan that resolves the problem. The solution should be factored out from the portion of the system that provides functionality.

Every DRTS needs a method for managing and allocating resources in a way that allows real-time (RT) constraints to be met; it is often the case that the techniques needed to do this are re-invented each time a new system is engineered, thus increasing the cost of building such systems.

Simply recording the time taken to execute a portion of a DRTS is not enough to constitute adaptive behavior; some type of analysis is needed. A mechanism for making decisions about the data received is required in order to determine whether the DRTS is performing satisfactorily. Further, if performance is less than desired, the source of the performance lag must be identified and remedied. If a performance constraint is violated, what can be done to restore the performance to the required level, while not causing violations of other performance constraints? For example, on the destroyer, would it improve the performance of the tracking process to move that process to a different host? This is the type of question that needs answered.

Create an algorithm that can diagnose performance and workload data pertaining to the DRTS (see the Resource and QoS Monitor pattern in section 3.2), discover and analyze alternative allocations, and adapt to the allocation to the current needs of the DRTS. The Allocation Manager pattern’s main role is to coordinate information received, determine if the current allocation is acceptable, and determine if another allocation would be better. For instance, if Allocation Manager decides that a DRTS is violating its performance requirements, some of the options available for restoring the needed performance include relocating or replicating the violating process. Acting on a chosen option is the realization of the adaptation, which requires the Resource Allocation Controller pattern (see section 3.3).

Examples of this pattern are found in [5], [12], and [13]. A diagram depicting this pattern in the DeSiDeRaTa system [13] is given in figure 1.

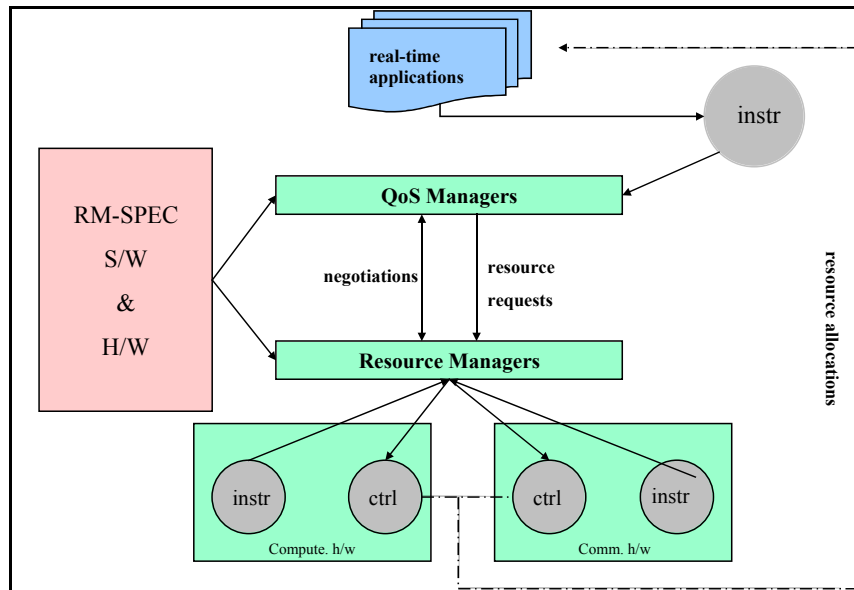


Figure 1: A system containing an allocation manager (from [13]).



This pattern answers the question of how to adapt. The Resource and QoS Monitor pattern (3.2) and the Resource Allocation Controller pattern (3.3) are needed to complete this pattern. The Resource and QoS Monitor pattern provides to the Allocation Manager information to analyze, and the Resource Allocation Controller pattern provides to the Allocation Manager a means of executing its analysis. Additionally, to effectively utilize this pattern, a DRTS needs to be engineered with the System Specification, System Profile and QoS Information Reporter (4.1-4.3).

3.2 Resource and QoS Monitor**



A DRTS must have the ability to monitor its performance and resource needs. Furthermore, there must be some way to determine the status and utilization of the resources. Otherwise, how can the system determine if its deadlines and resource needs are being met, and how can it make informed resource allocation decisions? This pattern helps to complete the Allocation Manager pattern.



The performance data of a DRTS is a crucial element for determining whether or not the system is meeting its real-time requirements. Additionally, the resource needs of a DRTS and the utilization levels of the resources determine the adequacy of a particular allocation.

We want to be able to determine the performance and resource needs of the DRTS and the utilization levels of the resources. The performance metrics may include end-to-end latency and workload for a path through a DRTS. Additionally, the information about the resources may include resource status (off/on) and utilization, as well as several device-type-specific attributes, including context switching rate and free memory for host resources, and expected latency, available bandwidth, and collisions for network resources. For the destroyer, we might record the time it takes to engage a threat track from the time it is first detected, and to determine the CPU utilization for processing a single track.

Provide a mechanism for gathering information regarding QoS and workload information from the processes of the DRTS. This can be accomplished by inserting reporting mechanisms into the DRTS (see the QoS Information Reporter pattern in section 4.3) and by developing a component to receive and process the reported information. Some metrics may also be obtained by examining operating system tables. The processing of the information may require aggregation of many process-level events into a single QoS metric. For periodic and distributed systems, this aggregation can be simplified by tagging and timestamping event messages.

Additionally, provide a means of obtaining information about the resources and of tracking the historical utilization values and trends for the resources. Provide aggregate load indices for resources, such as a host load index that combines normalized values for CPU utilization, context switching rate and available memory.

Develop an algorithm to determine when constraint thresholds are crossed and to request a reallocation of resources. Example constraint thresholds include real-time deadlines and

resource utilization thresholds (such as the threshold that is used in rate monotonic analysis [14]).

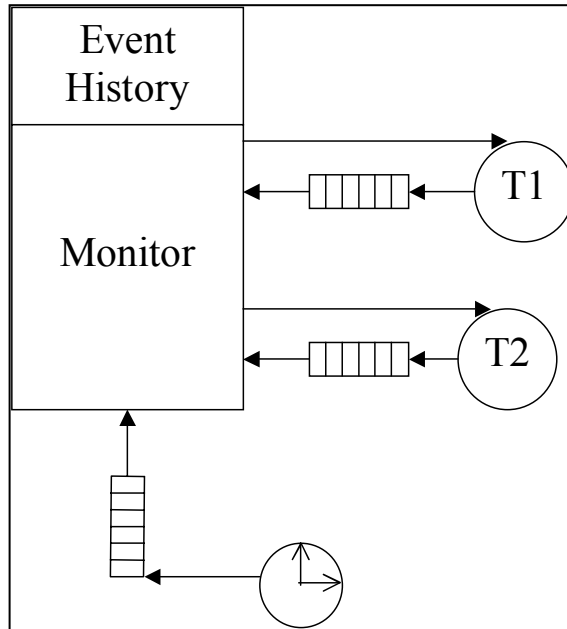


Figure 2: A resource and QoS Monitor (from [6]).

Examples of this pattern appear in [5], [6], [8], [15], [16], [18], [19], [30], and [31]. Figure 2, from [6], exemplifies this pattern.



This pattern answers the question of knowing when to adapt. The QoS Information Reporter (4.3) can be used by this pattern for identifying fluctuations in DRTS needs and performance.

3.3 Resource Allocation Controller**



The Allocation Manager pattern requires a means of executing its decisions in order to achieve adaptive behavior. This pattern provides the necessary means.



Mechanisms for initiating adaptations must exist if DRTSs are to change their behavior in response to conditions caused by dynamic environments.

Adaptation, by definition, means adjusting behavior in response to conditions in the environment. Some type of mechanism must be built into the DRTS so that it can adjust to the dynamic environment in which it is executing. This is an important piece of the puzzle, providing a way to carry out the decisions made at the analysis level (see Allocation Manager Pattern, 3.1). For example, on a ship, if the real-time process responsible for evaluating tracks becomes overloaded, and it is determined that the process should be migrated to a more powerful host, then there must be a mechanism for performing process migration [4].

Provide methods for controlling the DRTS. For example, provide mechanisms for process migration and replication; this can be implemented most efficiently within an operating system, but may also be implemented using common process control services. Another common mechanism is to adjust the resource usage of the DRTS by changing the fidelity of one or more of its algorithms; this can be accomplished by having the DRTS provide a control interface, as in DQM [15].

This pattern occurs in [4], [12], [13], [15], [16], [18], and [21]. Figure 3 is an example from [21].

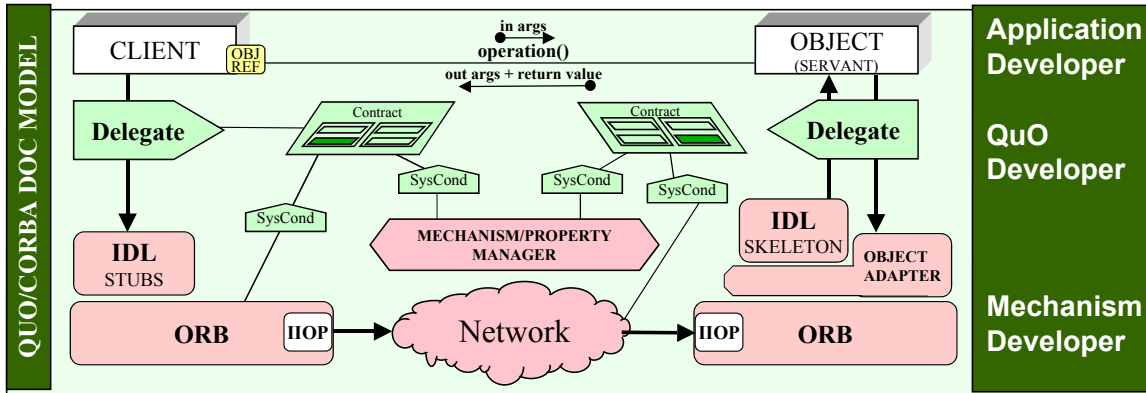


Figure 3: A system containing an application controller (from [21]).



A manner for controlling the allocation of resources is crucial for providing adaptive behavior to the system. This pattern completes the Allocation Manager pattern, and it answers the question of how to achieve adaptive behavior.

4 Engineering Dynamic Real-time Systems

The patterns presented in the previous section have shown how to achieve adaptive behavior. However, how does one build a DRTS that can utilize the solutions provided by these patterns?

4.1 System Specification*



Each DRTS has various properties, such as execution time and QoS requirements, as we have discussed in previous patterns. How are the properties of a DRTS described?



The Allocation Manager needs to be informed about the characteristics and requirements of the DRTS that it needs to manage. Furthermore, the descriptions of the QoS-relevant features of a DRTS should be factored out from the components that implement the functionality of the application, to facilitate ease of maintenance and development.

To be an intelligently adaptive software system, the manager of the resources on which the system executes must be informed of relevant properties of the software system. The problem lies in determining how to communicate to the manager how the system is organized, the connectivity among its constituent parts, and/or its runtime constraints. Further, if there are runtime constraints, to what part or parts of the system do these apply and, furthermore, how are these parts related? How does one incorporate workload-dependent execution times? In the example of the destroyer, the system specification may include the deadline for the track review processing loop, and the resource usage profiles of the loop at various workloads.

Develop a format for describing this information for the real-time system. This format must be able to accurately and unambiguously represent the communication and precedence relations between applications of the DRTS. Also, each application within the system may need to provide information about where each application “lives,” such as on which resource it resides (if there is more than one possibility), the path to the executable on that resource, any environment variables it accesses, as well as any arguments needed to successfully start the executable. Furthermore, information about replication, relocation, importance, QoS requirements, and resource usage profiles needs to be detailed.

```

<proposal>
  <mode>
    <or>
      <ci name="radioVHF" state="onLine" />
      <ci name="radioUHF" state="onLine" />
    </or>
  </mode>
  <QoS type="latency">
    <upperPoint secs="1.0" prob="0.99" />
    <upperPoint secs="4.0" prob="0.9999" />
  </QoS>
  <load type="interMessageTime">
    <upperPoint secs="1.0" prob="0.0001" />
    <upperPoint secs="1.0" prob="0.9999" />
  </load>
  <load type="messageSize">
    <upperPoint bytes="256" prob="1.0" />
    <upperPoint bytes="32" prob="0.5" />
  </load>
  <load type="priority">
    <urgency val="10" />
    <importance val="2" />
  </load>
</proposal>

```

Figure 4: A system specification (from [17]).

Examples of this pattern are found in [5], [6], [13], [17], [20], [22], [23], and [24]. Figure 4 gives an example of a system specification from [17].



The information about a particular DRTS is important for management of the DRTS. The question still remains about how to determine how much host and network resources will be used on a particular process. The System Profile pattern answers this question.

4.2 System Profile*



Real-time systems that adapt to dynamic environments may process varying workloads. The resource usage patterns at various workloads should be characterized, to enable effective online allocation analysis.



Ensuring DRTS requirements are met is a difficult task if there is no way to determine the amount of processing and communication required to operate in a particular environmental context.

The resource usage of a DRTS changes, depending on its workload. The execution time and communication amount of a DRTS depends on the amount of work it has to perform. One estimated execution time value, say, an average case, generally is inadequate for accurately characterizing the amount of resources the application will need at any given workload [7]-[11]. So, this execution time must be computed as a function of the size of the workload. Further, to perform effective schedulability analysis and resource allocation, sufficient information must be provided so that this number can be efficiently and accurately assessed during the execution of the application.

Generate resource usage profiles by gathering resource usage and processing time information at different workloads. First, determine a range of workloads (e.g., set the minimum equal to one, and the maximum equal to the workload that causes the host on which profiling is being performed to have 70% CPU utilization). Next, determine the minimum sample size that meets error and accuracy requirements (e.g., use the statistical Z-test [32]). Then, collect N profile samples at the determined workload. Finally, determine if enough profile points have been gathered to accurately predict execution time for this workload in the specified range (e.g., perform prediction by using piecewise linear interpolation between points and check errors of the predictions). If not, divide the interval in half, and gather N profile samples at the midpoint of each subinterval. This process is explained fully in [3].

Figure 5 depicts the solution presented in [3] as a flow chart. Other examples can be found in [25]-[29].

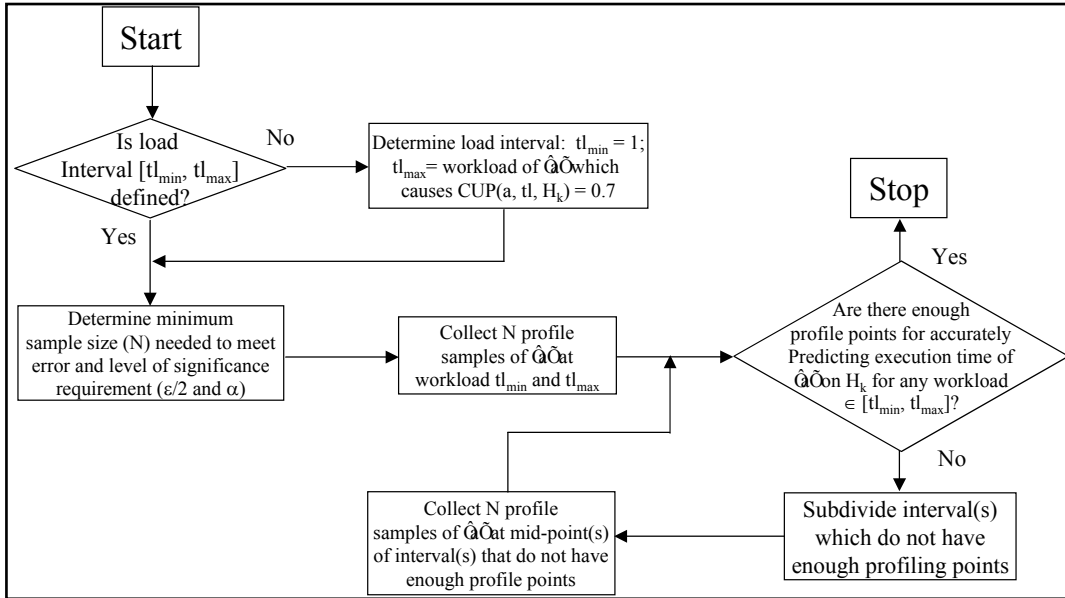
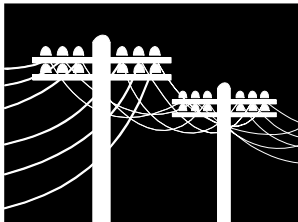


Figure 5: A system profiler (from [3]).



Now, we have a method for maintaining all necessary information about a given DRTS. The last piece of the puzzle concerns the dynamic reporting the needs of the DRTS, solved with the QoS Information Reporter.

4.3 QoS Information Reporter**



A characteristic that makes a DRTS system dynamic is the fact that the needs of the system change during the course of execution.



How does an application make its resource needs and QoS-relevant performance metrics known? The changing needs must be made known.

The current resource needs and current QoS of a DRTS need to be known. Without this information, especially information about a change in needs, it is impossible to create an adaptive solution for DRTSs. If the cruiser suddenly comes under attack, the response to this attack, from a resource management perspective, must be immediate. The change in resource needs and usage will be dramatic. The allocation manager must be able to have a way to incorporate this type of information into its decision making process.

Provide a method for gathering and communicating the CPU utilization, communication needs, and memory usage for each application, and for measuring the amount of time taken to process data. Report the workload (e.g., number of tracks) and/or resource usage. Also, take timestamps at the beginning and end of each cycle, report the cycle number, and record the event (e.g., start, end). With this information, it can be determined how long processing takes for one unit of data (e.g., one track) to progress through the system.

An example of this pattern can be seen in the DQM [15] system architecture diagram, which is shown in figure 6. Other examples of this pattern are found in [5] and [13].

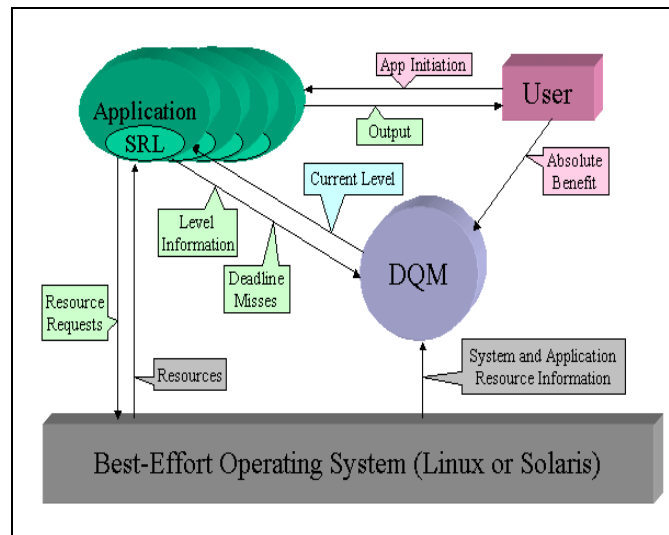


Figure 6: A system that contains a QoS information reporter (from [5]).



This pattern provides information needed by the Allocation Manager to make intelligent decisions and analyses of alternative allocations for real-time processes, by allowing the DRTS communicate its needs and performance.

5 Summary

This paper has presented two groups of patterns: a set for designing reusable middleware components for resource management, and a set for engineering systems that employ middleware services. These two sets work together to provide a complete solution for designing a dynamic real-time system that can adapt to changes in its environment. The Allocation Manager pattern acts as the brain, collecting information from the Resource and QoS Monitor, and providing reallocation actions to be executed through the Resource Allocation Controller. The System Specification provides characteristics and needs of the DRTS as well as System Profiles to the Allocation Manager. Finally, the QoS Information Reporter solves the problem of communicating changes in DRTS needs and performance.

6 References

- [1] C. Alexander, S. Ishikawa, M. Silverstein, et al., *A Pattern Language*, Oxford Univ. Press, New York, N.Y., 1977.
- [2] J.P. Loyall, P. Rubel, R. Schantz, et al., "Emerging Patterns in Adaptive, Distributed Real-Time, Embedded Middleware," *Proc. 9th Conf. Pattern Languages of Programming (PLOP 02)*, <http://jerry.cs.uiuc.edu/~plop>.

- [3] Y. Zhou, L.R. Welch, E. Huh, et al., "Execution Time Analysis for Dynamic, Periodic Processes," *Proc. 9th Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, April 2001.
- [4] D.S. Milojicic et al., "Process Migration," *ACM Computing Surveys*, vol. 32, no. 3, Sept. 2000, 241-299.
- [5] J.P. Loyall et al., "Specifying and Measuring Quality of Service in Distributed Object Systems," *Proc. of the 1st Int'l Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '98)*, 20-22 April 1998, Kyoto, Japan.
- [6] F. Jahanian, "Run-Time Monitoring of Real-Time Systems," *Advances in Real-Time Systems*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1995.
- [7] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proc. of the 19th IEEE Real-Time Systems Symposium*, 1998, 3-13.
- [8] D. Haban and K.G. Shin, "Applications of real-time monitoring for scheduling tasks with random execution times," *IEEE Trans. Soft. Eng.*, **16(12)**, Dec. 1990, 1374-1389.
- [9] J. Lehoczky, "Real-time queuing theory," in *Proceedings of the 17th IEEE Real-Time Systems Symposium*, 186-195, IEEE Computer Society Press, 1996.
- [10] D.B. Stewart and P.K. Khosla, "Mechanisms for detecting and handling timing errors," *Communications of the ACM*, **40(1)**, January 1997, 87-93.
- [11] T.S. Tia, Z. Deng, M. Shankar, et al., "Probabilistic performance guarantees for real-time tasks with varying computation times," *Proc. 1st IEEE Real-Time Tech. and Apps. Symp.*, 1995.
- [12] M. Litzkow, M. Livney, and M. Mukta, "Condor: A hunter of idle workstations," *Proc. 8th International Conference on Distributed Computing Systems*, San Jose, June 1998, 104-111.
- [13] B. Ravindran, L.R. Welch, and B. A. Shirazi, "Resource Management Middleware for Dynamic, Dependable, Real-Time Systems," *The Journal of Real-Time Systems*, 20:183-196, Kluwer Academic Press, 2000.
- [14] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, **20**, 1973, 46-61.
- [15] S.Brandt, et al., "A dynamic quality of service middleware agent for mediating application resource usage," *Proc. 19th IEEE Real-Time Systems Symposium*, 307-317, IEEE Computer Society, Los Alamitos, CA, 1998.
- [16] B. Shirazi, A. Hurson, and K. Kavi, "Tutorial on Scheduling and Load Balancing," IEEE Computer Society Press, 1995.
- [17] J. Cross and D. Schmidt, "Quality Connector: A Pattern Language for Provisioning and Managing Quality-Constrained Services in Distributed Real-time and Embedded Systems," *Proc. 9th Conf. Pattern Languages of Programming (PLOP 02)*, <http://jerry.cs.uiuc.edu/~plop>.

- [18] E. Huh and L.R. Welch, "An Efficient Schedulability Analysis Policing Technique for Periodic, Dynamic, Real-Time Applications," *The 10th Workshop on Parallel and Distributed Real-Time Systems*, April 2002.
- [19] C. Cavanaugh, *Quality of Service Management for Dynamic, Distributed Real-time Systems*, doctoral dissertation, Dept. of Computer Science and Engineering, University of Texas, Arlington, 2000.
- [20] K.B. Kinny and K. J. Lin, "Building flexible real-time systems using the Flex language," *IEEE Computer Society Press*, vol. 24, no. 5, May 1991, 70-78.
- [21] C. Rodrigues, "Using Quality Objects (QuO) Middleware for QoS Control of Video Streams," *OMG Embedded and Real-Time Distributed Object Systems Workshop*, January 7-10, 2002, Burlingame, CA.
- [22] V. Nirkhe, S.K. Tripathi, and A. Agrawala, "Language support for the MARUTI real-time system," *Proc. Of the IEEE Real-Time Systems Symposium (RTSS)*, December 1990, 257-266.
- [23] Y. Ishikawa, H. Tokuda, and C. Mercer, "Object-oriented real-time language design: Constructs for timing constraints," *Proc. of the Object-Oriented Programming Systems, Languages, and Applications (ECOOP/OOPSLA'90)*, 1990, 289-298.
- [24] K. Takashio and M. Tokoro, "DROL: An object-oriented programming language for distributed real-time systems," *Proc. of the Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'92)*, 1992, 276-294.
- [25] U. Chandra and M.G. Harmon, "Predictability of Program Execution Times on Superscalar Pipelined Architectures," *Proc. 3rd Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'95)*, 25 April 1995, 104-112.
- [26] A.K. Mok, "Evaluating Tight Execution Time Bounds of Programs by Annotations," *Sixth IEEE Workshop on Real-Time Operating Systems and Software*, Pittsburgh, PA, 75-80.
- [27] C.Y. Park and A.C. Shaw, "A Source-Level Tool for Predicting Deterministic Execution Time of Programs," *Technical Report 89-09-12*, Department of Computer Science and Engineering, University of Washington, Seattle, Washington, 1989.
- [28] P. Puschner and A. V. Schedl, "Computing maximum task execution times - a graph based approach," *Proc. of IEEE Real-Time Systems Symposium*, vol. 13, Kluwer Academic Publishers, 1997, 67-91.
- [29] F. Burns, A. Koelmans, and A. Yakovlev, "WCET Analysis of Superscalar Processors Using Simulation With Coloured Petri Nets," *The International Journal of Time-Critical Computing Systems*, vol.18, Kluwer Academic Publishers, 2000, 275-288.
- [30] Robert Hill, Balaji Srinivasan, Shyamalan Pather, et al., *Temporal resolution and real-time extensions to linux*. Technical Report ITTC-FY98-TR-11510-03, Information and Telecommunication Technology Center, Department of Electrical Engineering and Computer Sciences, University of Kansas, June 1998.

[31] A. Barak and O. La'adan, "The MOSIX Multicomputer Operating System for High Performance Cluster Computing," *Journal of Future Generation Computer Systems*, 13(4-5), March 1998, 361-372.

[32] J.L.Devore, *Probability and Statistics for Engineering and the Sciences*, Brooks/Cole Publishing Co., 1982.