

JAVAMAC AND RUNTIME MONITORING FOR GEOINFORMATICS GRID SERVICES

Ann Q. Gates, Steve Roach, Irbis Gallegos, and Omar Ochoa
Department of Computer Science
The University of Texas at El Paso

Oleg Sokolsky
Department of Computer and Information Sciences
University of Pennsylvania

1 Introduction

A rising trend in the software industry is in distributive, collaborative problem-solving, which has been made possible by grid and web service technology. The grid is characterized as an open system in which users, software components, and computational resources (all owned by different entities) come and go on a continuous basis. Grid applications are built as collections of interacting processes that communicate over a network and support a high degree of automation, providing flexible collaborations and computation on a global scale. This paper describes ongoing research in software runtime monitoring using the JavaMaC system and preliminary ideas on the application of runtime monitoring to establish assurance of grid applications.

1.1 Motivation

Many grid applications require the integration of data and knowledge from multiple sources and domains. In addition, there are aspects of these applications that make them complex. The processes need to synchronize and, thus, they exhibit time-dependent behaviors. Because of the needed integration of components, software must interact with physical devices and other software over external networked environments. In distributed and real-time systems, it is difficult to determine if timing and computational properties are achieved due to uncertainty of environments external to the system and errors in specifications or implementations.

The application of the research described in this paper is on the GEO-science Network (GEON), an NSF collaborative effort among researchers from 13 universities representing a broad cross section of information technology and earth science disciplines. Partner institutions include San Diego Supercomputer Center, Arizona State University, Cornell, Virginia Tech, and UTEP. GEON is building digital libraries of high-quality geological information and integrated software tools for data access, analysis, modeling, and visualization, providing resources for researchers, students, teachers, and the public. Other examples of applications that use grid technologies to support and enhance the scientific process include Network for Earthquake Engineering Simulation System Integration (NEESgrid) and e-Science.

A requirement of the GEON project is to provide the scientist with assurance that the data and computations are correct. To elucidate example uses of GEON, we present two scenarios. The first scenario presents a sequence that is achievable in the near term by adapting state-of-the-art tools. The second scenario presents a vision of the future. The scenarios are not intended to be domain specific; however, they describe practices to which the grid aspires and suggest needed research.

Scenario 1: A developer creates an application for a scientist using a grid service development toolkit. The developer discovers available services that meet his or her needs by querying a registry. The information provided to the developer includes such things as: signature; port bindings for the service provider; names, data types and descriptions of the state information that the developers have exposed about the implementation of the service; and properties about the service. The developer and scientist, who have limited to no knowledge of formal specification, use a tool to elucidate the properties. Each chooses an approach for visualization that best suits his or her model of understanding. The developer creates the application by invoking various services and integrating the results. In addition, the tool helps the scientist define properties that ensure that the application as a whole behaves as needed. The application is monitored during runtime with the monitor running on the service provider.

Scenario 2: A workflow stored on the grid defines a network of tasks for satisfying a scientist's request. When initiated, the workflow is forwarded to an autonomous grid agent that identifies registered services that satisfy the tasks in the workflow. The agent schedules services in an efficient manner utilizing information in the workflow and current meta-data associated with the service. The workflow is annotated with properties that are used by a monitor to verify that the services uphold their contracts. In this scenario, only the interfaces between services are exposed. If the monitor discovers that a service does not conform to the requirements, it notifies the agent, and the agent identifies a different service to satisfy the task. The monitor updates the registry so that other agents using the registry are better able to satisfy similar requests in the future.

1.2 Focus of Paper

In grid computing, users compose their applications from services. It is not enough to know the static interface of the service (i.e., the types of its inputs and outputs) in order to ensure that combined services produce the correct result. One also should consider the services' timing parameters and how the services produce inputs from outputs. Such semantic properties are important not only at design time, when the services are composed, but also at run time when the

services are being used. At design time, it is necessary to check whether the composition of the properties of the services will imply the desired property of the application. At run time, it is necessary to determine whether each service is fulfilling its promises. One of the goals of grid computing is to design solutions dynamically by searching for services at run time. In this situation, it is not feasible to test components prior to using them in the design of a solution. Monitoring application behavior at runtime enables the dynamic replacement of misbehaving services or, if a service offers a reconfiguration interface, the adjustment of parameters to achieve the desired behavior.

In order to check whether the property holds at run time, we need to be able to monitor the relevant aspects of the service behavior. What aspects are relevant is dependent on the property and usually can be inferred by inspecting the primitive components (events and state) in the property specification. Services in the grid, however, are available to the user as black boxes. They can be implemented in a variety of ways using different programming languages, and they can run on different kinds of platforms. Monitoring the relevant aspects of the service behavior at run time makes it possible to check whether the property holds. Instrumenting services for monitoring by the user is infeasible. Services, therefore, must provide an interface that will allow checking of the properties by an external monitor.

This paper outlines an infrastructure for specifying properties of services and applications, an interface for exposing relevant behaviors of a service, and a “meta-service” for run-time checking of service properties. In addition to these topics, the paper describes challenges in monitoring grid services and discusses related work. The paper closes with a summary.

2 Monitoring Grid Services

This section examines the focus of the research concerning the requirements for monitoring properties. In addition, it looks at an existing runtime monitoring tool that is being adapted to monitor grid services.

2.1 Monitoring Requirements

To effectively monitor services for conformance to required behavior, it is necessary to identify the properties that separate conforming services from nonconforming services. Two questions are being addressed by the research in this area. First, what are the requirements for monitoring the integrity of data and computational properties of grid services, i.e., properties that demonstrate that the service is behaving correctly? Second, what support is required for monitoring the integrity of grid services?

The ability to monitor services for data and computational integrity requires: 1) the definition of a behavioral interface for grid services such that each service will advertise a set of properties that the service guarantees; 2) the capability to browse these properties during the application design and select the ones that are important for the application; and 3) a set of hooks in the service that allows implementation-independent access for events and state variables that are necessary for evaluating the property.

The Web Service Description Language (WSDL) [FK02, CC01] document defines the interfaces of services that may be invoked by other entities as well as other capabilities of the web service. It is not standard practice today to provide descriptions of properties that a service guarantees to maintain. WSDL can be extended to provide this information. A service client would have the option to select the properties to be enforced. A service provider would have the ability to expose internal variables and types that she or he believes will be of interest to a client. This will allow clients of the service the ability to check intermediate results and loosen or strengthen properties, depending on how the service will be used.

Web Service Resource Framework (WSRF), which replaced the Open Grid Services Infrastructure (OGSI), defines specifications for accessing stateful resources. The resource framework includes WS-Notification [FF04] and WS-ResourceProperties [GC04, CA04]. Notification mechanisms are a viable approach for providing hooks in services. The goals of WS-Notification are to standardize the concepts, message exchanges, and WSDL and XML schema requirements for inter-object communication. The protocol will provide the monitor with the ability to register dynamically with the web services to be monitored and to specify the trigger conditions and events of interest. The goals of WS-ResourceProperties are similar to those of WS-Notification except for its application to resource projection, its association with the web service interface, and the messages defining the query and update capability against the properties of WS-Resource. It is essential that the conventions established by WS-Notification, WS-ResourceProperties, and other specifications provide dynamic support and access to the state of the computation.

2.2 MaC: a Runtime Monitoring Tool

A way to bridge the gap between high-level design and implementation is through the use of a formally defined monitoring scheme where the system is instrumented to monitor key aspects of its behavior and the monitored behavior is tested for conformance with the behavior of a correct system. One approach is the Monitoring and Checking (MaC) framework [KV99]. This research

is focused on how to monitor runtime behavior of systems that interact with external environments and the adjustments needed to build and apply tools to monitor grid applications.

The MaC framework allows users to specify system states to be monitored, define high-level events based on run-time system states, and describe correctness properties in terms of high-level events. Given these user defined artifacts, the MaC prototype automatically instruments the system with a filter and generates an event recognizer and a checker. At run-time, when a monitored state is reached, the filter passes this information to the event recognizer. The event recognizer uses the information to recognize high-level events, which are then sent to the checker. The checker determines whether the stream of high-level events conforms to the correctness properties.

Figure 1 shows the overall structure of the MaC framework. The framework includes two main phases: static phase and dynamic phase. During the static phase, i.e., before a target program is executed, run-time components such as a filter, an event recognizer, and a run-time checker are generated from a target program and a formal requirements specification. During the dynamic phase, the instrumented target program is executed while being monitored and checked with respect to the requirements specification.

The Meta-Event Definition Language (MEDL) is used to express requirements. It is based on an extension of a linear-time temporal logic. It can be used to express a large subset of safety properties of systems, including real-time properties. We use events and conditions to capture and reason about temporal behavior and data behavior of the target program execution; events are abstract representations of time and conditions are abstract representations of data.

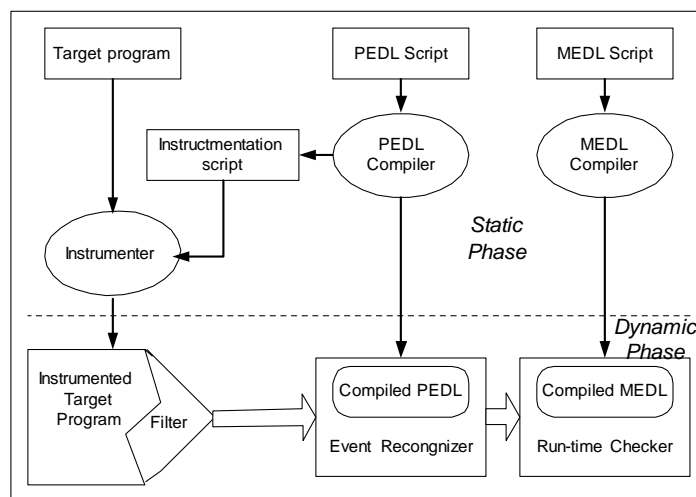


Figure 1. The structure of the MaC framework.

Requirements written in MEDL are expressed in terms of high-level events and conditions. Events occur instantaneously during execution, whereas conditions represent information that holds for duration of time. For example, an event denoting return from method *RaiseGate* occurs at the instant the control returns from the method, while a condition (*TrainPosition = 2*) holds as long as the variable *TrainPosition* does not change its value from 2. The distinction between events and conditions is important in terms of what the monitor can infer about the execution based on the information it gets from the filter. The monitor can conclude that an event does not occur at any moment except when it receives an update from the filter. By contrast, once the monitor receives a message from the filter that *TrainPosition* has been assigned the value 2, it can conclude that *TrainPosition* retains this value until the next update.

In addition to the requirements written in MEDL, a monitoring script relates these events and conditions with low-level data manipulated by the system at run-time. Monitoring scripts are expressed in the Primitive Event Definition Language (PEDL). PEDL is used to define what information is sent from the filter to the event recognizer, and how it is transformed into events used in high-level specification by the event recognizer. Based on the monitoring script, the system is automatically instrumented to deliver the monitored data to the event recognizer at run-time.

The reason for keeping the monitoring script (PEDL) distinct from the requirements specification (MEDL) is to maintain a clean separation between the system itself, implemented in a certain way, and high-level system requirements, independent of a particular implementation. PEDL, therefore, is tied to the implementation language of the monitored system in the use of object names and types. MEDL is independent of the monitored system. The separation between PEDL and MEDL ensures that the architecture is portable to different implementation languages and specification formalisms. Note that implementation-dependent event recognition insulates the requirement checker from the low-level details of the system implementation. This separation also allows us to perform monitoring of heterogeneous distributed systems, a characteristic of grid services. A separate event recognizer may be supplied for each module in such system.

A filter is a collection of probes inserted into the target program. The essential functionality of a filter is to keep track of changes of monitored objects and send pertinent state information to the event recognizer. An event recognizer detects an event from the state information received from the filter. Events are recognized according to a low-level specification. Recognized events are sent to the run-time checker. A run-time checker determines whether or not the current execution history satisfies a requirement specification. The execution history is captured from a sequence of events sent by the event recognizer.

To demonstrate the effectiveness of the MaC framework, we have implemented a MaC prototype for Java programs, called Java-MaC. Java-MaC targets Java executable code (i.e., bytecode). It is easy to deploy Java-MaC, because it automatically instruments the target program and generates the run-time components of Java-MaC based on requirements specifications written in two scripting languages, MEDL and PEDL-for-Java. The system is available at www.cis.upenn.edu/~rtg/mac.

2.3 MaC Extensions for Grid Monitoring

The current research efforts are focused on developing a fundamental understanding of timing property checking. The current MaC system only supports a subset of real-time properties. The work is being extended to consider the following categories of timing properties:

1. *Temporal dependencies* define the temporal relations among events. An example property is one that requires process A to complete before process B begins.
2. *Time constraints* restrict the time that an event is initiated or completed. For example, a system may require that a process is initiated at a given time, or that the difference in time when two events complete is within a specified interval.
3. *Rates* specify the number of particular events per unit of time, where the rates can be expressed as an exact value, or within an error range or percentage. For instance, an example rate property states that the frame rate of a video streaming application must be greater than a specified value, or that a node on a grid service must process ten data requests per second.

The MaC specification language is being extended to include a larger set of real-time properties, including the properties described above. The proposed extension will add regular expressions to simplify the specification for temporal dependencies and add timers and auxiliary timing functions to aid specifying timing constraints and rates. Regular expressions are known to add expressive power to the LTL [AF02] whereas the timers can catch the error as soon as it occurs. The major reason for the lack of timing properties in the current MaC implementation is that Java does not support real-time threads whose scheduling can be controlled to meet timing requirements. To remedy this, we will port Java-MaC to use real-time Java threads and explore how to support more accurate detection of timing violations.

Java-MaC defines two attributes for each event: time and value. It assigns a timestamp to each event based on the clock of the monitored system. The timestamps enable reasoning about the timing properties of the system and the order of events. Ideally, the clocks of the monitored systems and the monitoring system would be synchronized; however, in a grid environment this is unlikely especially if the monitor is executing as a separate service.

The GEON grid, is comprised of computation clusters. Machines on the grid are currently geographically dispersed among sixteen participating sites each with a set of clusters locally administrated and connected with wide-area networks [RC04]. Figure 2 shows the GEON grid architecture and possible mounting points for the monitoring elements.

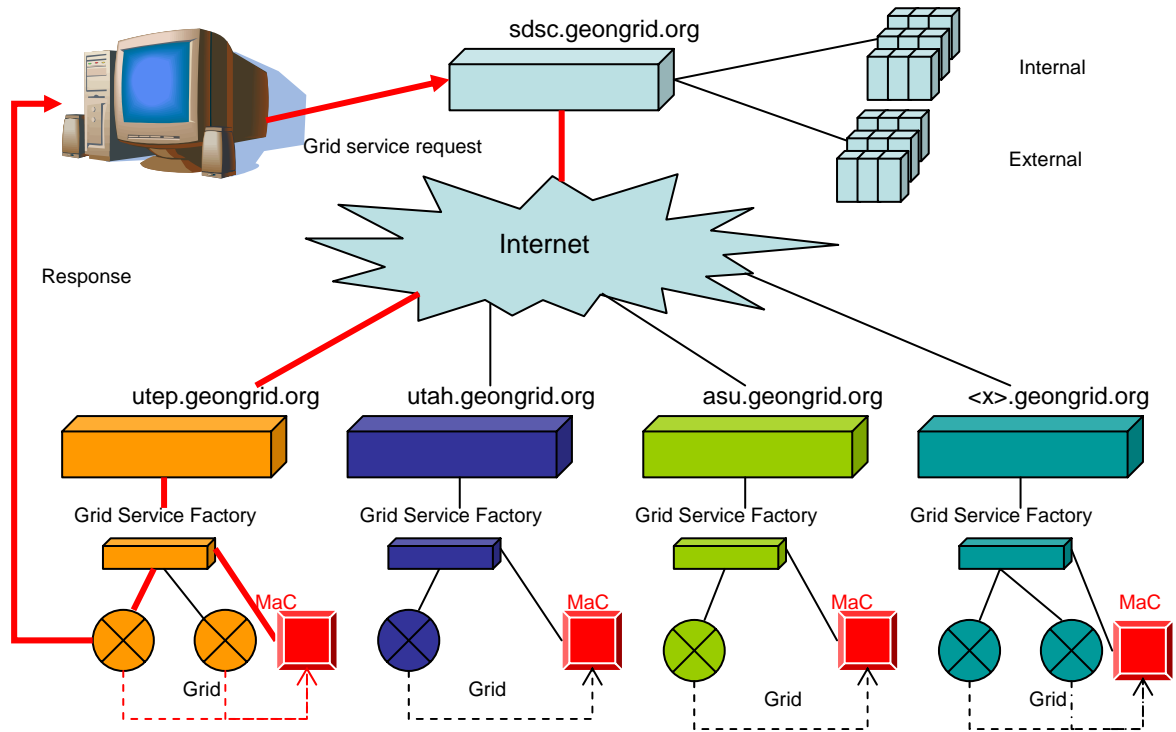


Fig 2. Example scenario of a grid service request satisfied by a node on the GEON grid and monitored using Java-MaC.

In the scenario presented in Figure 2, a grid service request is made to the GEON grid central node, which determines that the UTEP node can satisfy the request. A message is sent to the UTEP node to inform it of the requested service. Once acknowledged, the node initiates the Grid Service Factory and Java-MaC for monitoring the requested grid service. If a constraint violation is detected and a steering action is needed, Java-MaC will inform the Grid Service Factory, which determines if recovery is possible. The scenario in Figure 2 shows Java-MaC executing on each node in the grid; however, it may be advantageous for the monitor to run as a service itself, allowing Java-MaC to run on a node separated from the services.

In a grid computing environment, computational services run at different locations and communicate among themselves via message passing. Because connections among Java-Mac runtime components are established before the target application executes, the location of the grid service must be determined prior to its execution. This information is obtained and managed by

the grid management system. It determines the best communication medium, considering the communication overhead. For example, utilizing a communication medium such as TCP sockets require more overhead as compared to shared memory. Java-MaC provides three different communication mechanisms among the run-time components: TCP socket, FIFO file, and communication channels implemented by a user *InputStream* and *OutputStream* provided by the Java-MaC API. It is necessary to either create a new protocol that would allow efficient communication inside the grid by implementing a custom communication channel through Java-MaC's API, or to take advantage of local nodes on the grid for storing FIFO files and communicating through shared memory.

3 Related Work

The related work is divided into two sections. The first section examines general run-time verification techniques. The second section looks specifically at runtime monitoring of grid services.

3.1 Run-time Verification Techniques

Numerous researchers have studied run-time verification techniques. The MaC framework [KV99] provides an architecture to monitor and check a software system at run-time to ensure that the execution of a real-time system is consistent with its LTL-based requirements. It provides a prototype for software written in Java. JPax or Java Path Explorer [HR01] provides an environment for monitoring and verifying the execution of Java programs against specification based on temporal logic formulae. Time Rover [D03] monitors Java/C++ programs to check whether LTL requirement specification is violated or not. JASS (Java with ASSERTion) [BF01] is a precompiler that supports boolean assertions for Java. Jass takes Java source code and inserts pre/post conditions for methods and invariants for classes in special comments. The Java Run-time Timing constraint Monitor (JRTM) [ML97] aims to detect violation of timing properties in Java programs. JRTM uses Real-Time Logic (RTL) [AH92] as a requirement specification language. A Java program should be manually instrumented to put a probe in the place where a primitive event happens. Java Event Monitor (JEM) [ML99] is an event-mediator like the CORBA event channel. JEM receives predefined primitive events from event suppliers and detects composite events written in a Java Event Specification Language [LM98] based on these primitive events. Refer to [D02, D04] for a comprehensive comparison of run-time monitors.

3.2 Runtime Monitoring of Grid Services

There are several efforts that target monitoring of grid services. Many of them center on quality of service issues. The software oscilloscope project [GD04] targets the monitoring, tuning, and adaptation of grid applications, and the NetLogger toolkit collects data from software and hardware resources for *a posteriori* analysis [T04] of job performance, e.g., CPU loads, TCP transmissions, and I/O processing. Robinson [R04] is investigating monitoring of web services for quality of service. Deutsch et al. [DS04] verify sequences of inputs, states, and actions over data-driven web services provided by websites and based on interactions with users. A new standardization effort by the Global Grid Forum's Grid Resource Allocation Agreement Protocol (GRAAP) working group is defining a protocol called WS-Agreement to assist in resource discovery, inspection, and agreement. The protocol includes the ability to negotiate behavioral and negotiability constraints, e.g., the number of CPUs that are needed to run a particular service, and the number that can be negotiated.

4 Summary

The GEON project is a collaborative effort among numerous institutions, including UTEP, to create geoinformatics infrastructure. Because of the criticality of the grid services that will be deployed, scientists need assurance of the integrity of the data supplied by the service and assurance that the workflows and other scientific applications behave as expected. This paper describes the efforts toward developing a software runtime monitoring tool using the JavaMaC system and outlines the research questions and challenges that must be addressed to meet the requirements of the GEON community. In particular, the paper discusses the need for an interface that exposes state of a service, timing concerns, and communication issues. Although the work presented in the paper is presented in terms of the GEON project, it is applicable to all grid applications.

REFERENCES

- [AH92] Alur, R. and T. Henzinger, "Back to the Future: Towards a Theory of Timed Regular Languages," *IEEE Symposium on Foundations of Computer Science*, pp. 177-186, 1992. <http://citeseer.nj.nec.com/alur92back.html>
- [BF01] Bartetzko, D., Fischer, C, Möller, M., and H. Wehrheim, "Jass - Java with Assertions," *Electronic Notes in Theoretical Computer Science*, 55(2), Elsevier, K. Havelund and G. Rosu (eds.), 2001.
<http://www1.elsevier.com/gej-ng/31/29/23/83/33/28/55.2.002.pdf>

- [CC01] Christensen, E., Curbera, F., Meredith, G. and S. Weerawarana, "Web Services Definition Language (WSDL), Version 1.1", March 15, 2001.
<http://www.w3.org/TR/wsdl>
- [CA04] Czajkowski, K and A. Andrieux. [Globus Toolkit] Resource Management Overview, presented at the GlobusWorld Conference, January 2004.
www.globusworld.org/program/conference.asp.
- [D02] Delgado, N., "A Taxonomy of Dynamic Software-Fault Monitoring Tools," Master's Thesis, The University of Texas at El Paso, May 2002.
- [D04] Delgado, N., Gates, A., and S. Roach, "A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools," submitted to the *IEEE Transactions on Software Engineering*, 2003.
- [DJ04] DeRoure, D., Jennings, N., and N. Shadbolt, "The Semantic Grid: A Future e-Science Infrastructure," 2004.
<http://www.semanticgrid.org/documents/semgrid-journal/semgrid-journal.pdf>
- [DS04] Deutsch, A., Sui, L., and V. Vianu, "Specification and Verification of Data-driven Web Services, submitted to PODS 04. <http://www.cs.ucsd.edu/~lsui/publication/pods-final.pdf>
- [D03] Drusinsky, D., "Monitoring Temporal Logic Specifications Combined with Time Series Constraints," *Journal of Universal Computer Science*, 9(11), Nov. 2003.
http://www.jucs.org/jucs_9_11/monitoring_temporal_logic_specification/paper.pdf
- [FK02] Foster, I., Kesselman, C., Nick, J. and S. Tuecke, "The Physiology of the Grid, An Open Grid Services Architecture for Distributed Systems Integration", June 22, 2002.
<http://www.globus.org>
- [FF04] Foster, I., Frey, J., Graham, S., Tuecke, S., Czajowski, K., Ferguson, D., Leymann, F., Nally, M., Storey, T., Vambenepe, W., and S. Weerawarana, "Modeling Stateful Resources with Web Services," Version 1.0, January 2004.
<http://www-fp.globus.org/wsrif/ModelingState.pdf>
- [GD04] Gardner, M., Deng, W., Markham, T., Mendez, C., Feng, W., and D. Reed, "A High-Fidelity Software Oscilloscope for Globus," presented at the GlobusWorld Conference, January 2004. www.globusworld.org/program/conference.asp.

- [GC04] Graham, S., Czajkowski, K., Ferguson, D., Foster, I., Frey, J., Leymann, F., Maguire, T., Nagaratnam, N., Nally, M., Storey, T., Tuecke, S., Vambenepe, W., and S. Weerawarana, "Web Services Resource Properties (WS-ResourceProperties)," Version 1.0, January 2004.
<http://www-fp.globus.org/wsrf/WS-ResourceLifetime.pdf>
- [HR01] Havelund, K. and G. Rosu , "Monitoring Programs using Rewriting," in *Proc. Intl. Conf. ASE 01*, Nov. 2001, 145-144.
- [KV99] Kim, M., Viswanathan, M., Ben-Abdallah, H., Kannan, S., Lee, I., and O. Sokolsky, "Formally Specified Monitoring of Temporal Properties," *Proc. of the European Conference on Real-Time Systems - ECRTS'99*, pp. 114-121, Jun 1999,
<http://citeseer.nj.nec.com/kim99formally.html>
- [LM98] Liu, G., Mok, A., P. Konana, "A Unified Approach for Specifying Timing Constraints and Composite Events in Active Real-Time Database Systems," *Proc. of the IEEE Real-Time Technology and Applications Symposium*, June 1998.
<http://citeseer.nj.nec.com/liu98unified.html>
- [KK04] Kim, M., S. Kannan, I. Lee, O. Sokolsky, and M. Viswanathan, "Java-MaC: a Run-Time Assurance Approach for Java Programs," *Formal Methods in Systems Design*, to appear, 2004.
- [ML97] Mok, A. and G. Liu, "Early Detection of Timing Constraint Violation at Runtime," *Proc. of IEEE Real-Time Systems Symposium*, Dec. 1997.
<http://citeseer.nj.nec.com/mok97early.html>
- [ML99] Mok, A. and G. Liu, "Implementation of JEM - A Java Composite Event Package," *Proc. of the IEEE Real-Time Technology and Applications Symposium*, pp. 68-78, June 1999.
<http://csdl.computer.org/comp/proceedings/rtas/1999/0194/00/01940068abs.htm>
- [R04] Robinson, W., "Runtime Monitoring of Web Service Requirements," *Proceedings of 11th IEEE International Requirements Engineering Conference*, 2003, pp. 65-74.
- [RC04] Rocks Cluster Distribution: Award Winning Open Source High Performance Linux Cluster Solution, <http://rocks.npaci.edu/rocks>, November 30, 2004.
- [T04] Tierney, B., "Grid Troubleshooting with the NetLogger Toolkit," presented at the GlobusWorld Conference, January 2004. <http://dsd.lbl.gov/NetLogger/>.