OCM-G – Grid Application Monitoring System: Towards the First Prototype*

Bartosz Baliś[1,2], Marian Bubak[1,2], Tomasz Szepieniec[2], Roland Wismüller[3,4], and Marcin Radecki[2]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
[2] Academic Computer Center – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
[3] LRR-TUM – Technische Universität München, D-80290 München, Germany
[4] Institute for Software Sciences, University of Vienna, A-1090 Wien, Austria
{balis,bubak}@uci.agh.edu.pl, {t.szepieniec,m.radecki}@cyf-kr.edu.pl, wismuell@in.tum.de
phone: (+48 12) 617 39 64, *fax:* (+48 12) 633 80 54 *phone:* (+49 89) 289-28243

Abstract

This paper presents the OCM-G — an application monitoring system designed to support the development of applications in Grid environments. The OCM-G provides information, manipulation and event detection monitoring services, which are available via the standardized interface OMIS (On-line Monitoring Interface Specification). Tools connected to the OCM-G use the monitoring services it provides to collect information about applications and to manipulate them. The paper describes the architecture of the system, its interfaces, the request distribution mechanism, the start-up procedure as well as the functionality of the first prototype of the OCM-G and plans for future development.
Keywords: Grid, monitoring, services, interactive applications, OMIS, tools

1 Introduction

Monitoring of applications is important for tools, such as debuggers or performance analyzers, which are used by developers to support the application development [4]. Debuggers need such functionality as reading and writing a process's memory, starting/stopping the process, setting a breakpoint in a process, etc. Performance analyzers, on the other hand, need information about an application's execution, specifically, the amount of data sent by a process, the delay of synchronization/communication, etc. They use this kind of information to define performance measurements, for example: "measure total delay of sending messages by process p" or "measure the amount of data received by processes p1, p2, p3".

In this paper, we present a monitoring infrastructure designed to provide a variety of monitoring services for different kinds of tools, the OCM-G (Grid-enabled OMIS-compliant Monitoring system). The OCM-G is an autonomous

*This work was partly funded by the European Commission, project IST-2001-32243, CrossGrid [6]

structure, whose services are available via the standardized interface OMIS (On-line Monitoring Interface Specification) [11]. The OCM-G is Grid-enabled [7], which means that it is designed to address the high requirements for scalability and efficiency, it is able to properly start-up in a Grid environment, it ensures security, and it provides monitoring services which are useful and desired in the Grid.

There are different systems for grid-oriented monitoring [1]. The Globus Heartbeat Monitor [10] reports failures of processes and provides notification of process status exception events. Network Weather Service [16] monitors performance of networks and computational resources and forecasts their performance. The Information Power Grid project is developing monitoring infrastructure [14] built of sensors, actuators, and event services represented with XML. NetLogger [9] is designed for analysis of communication inside applications. GRM and PROVE [12] may be used for performance monitoring of applications running in distributed heterogeneous systems. Autopilot/Virtue [13] is a performance measurement and resource control system. The DataGrid project [5] is developing monitoring systems oriented towards applications which operate with large data files.

The paper is organized as follows. The following section presents the architecture of the OCM-G and the monitoring environment. Sections 3 and 4 describe the interfaces of the monitoring system. In the next three sections we give some more details about important parts of the design: request distribution (Section 5), processing of conditional requests (Section 6), and start-up of each component (Section 7). Finally, Section 8 describes the status of project and plans for the future.

2 The OCM-G and Monitoring Environment

The architecture of the OCM-G is shown in Fig. 1 The external interface of the OCM-G is OMIS (On-line Monitoring Interface Specification). OMIS defines the target system as a set of objects, and specifies a number of monitoring services to gather information, do manipulations, and detect events in these objects (see next section for an extended description). Also the internal communication between OCM-G components is based on OMIS.

The OCM-G is composed of two types of independently running components: Local Monitors (LM) and Service Managers (SM). Local Monitors are transient components of the OCM-G in the sense that they can be created and destroyed. LMs must run on each host, where there are application processes to be monitored. A LM's function is to accept OMIS requests and execute them. OMIS requests sent to a LM must pertain only to local application processes, i.e. those on the same node as the LM. Local Monitors run as user processes. Each Grid user has its own, "private" collection of LMs. This means that on one host there can be multiple Local Monitors if the host is used by multiple Grid users at the same time.

Service Managers are permanent components of the OCM-G. There is one

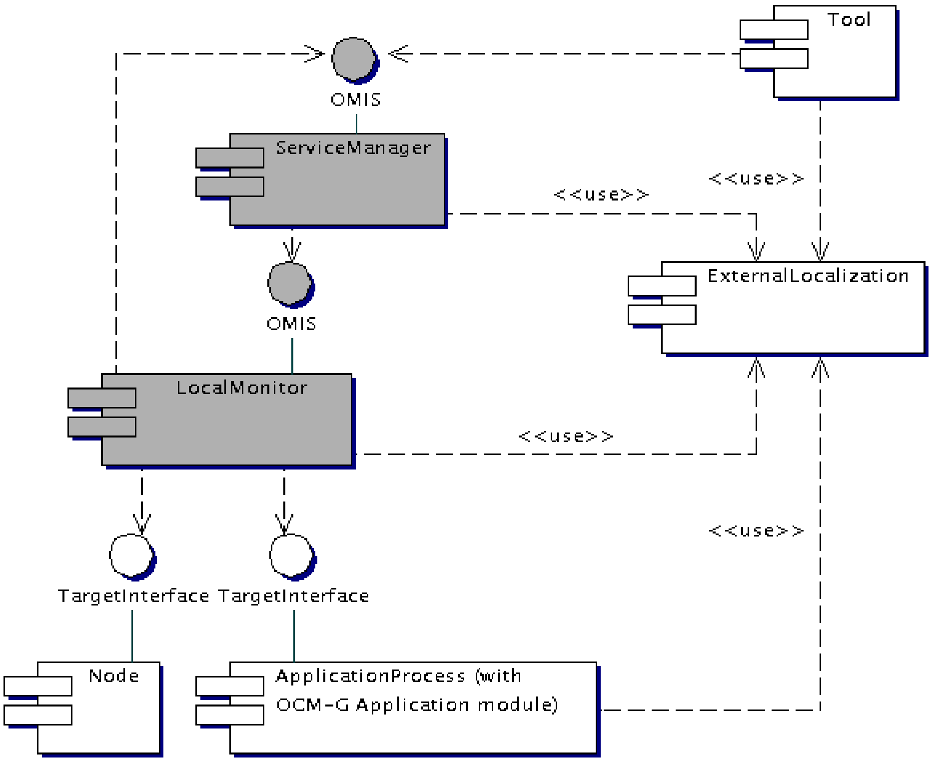

Fig. 1: OCM-G architecture

SM running per site of the Grid. A SM's function is to accept OMIS requests from a tool, to split them into partial requests for the Local Monitors, to collect replies from the LMs, to assemble them into a single reply, and to send the reply back to the tool. SMs run as unprivileged processes.

OCM-G also provides a library linked with the application processes (AP), which we call Application Module (AM). This library is responsible for initializing the monitoring in each application process and for performing monitoring actions that are executed within the application process context.

All components defined above use an External Localization to locate OCM-G parts distributed in the Grid. External Localization is a generalization of the special information service accessible in the environment. An example can be the Globus MDS or the shared file system on local clusters. In OCM-G there can be many different External Localizations mechanisms used at the same time. Information about localization of components taken from External Localization makes possible to form a monitoring environment, as shown in Fig. 2. The monitoring environment consists of one or more tools and application processes in addition to OCM-G.

A tool communicates with the OCM-G via a Service Manager to collect

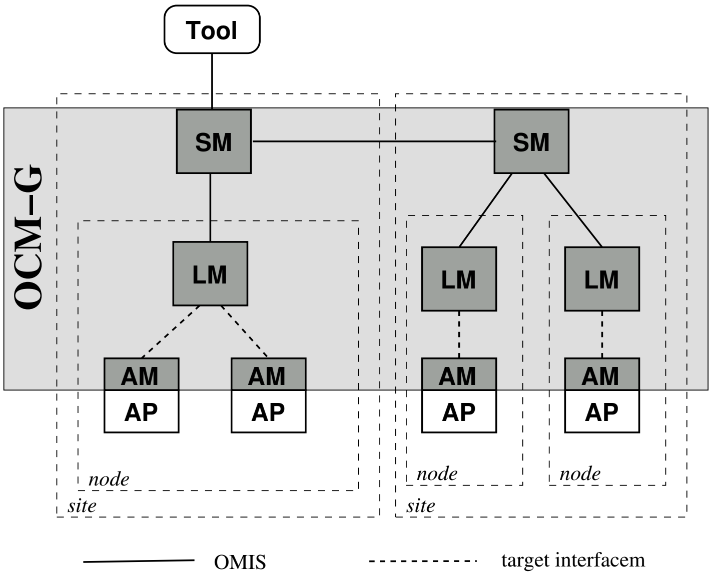

Fig. 2: Monitoring environment based on OCM-G

information about (and possibly manipulate) an application process. Every application process is under control of a Local Monitor which performs the actual actions on it, by a target interface explained in Section 4. The Service Manager accepts requests from the tool and forwards it to the Local Monitor. Vice versa, it collects the replies from the Local Monitor and returns them to the tool.

3 Monitoring Interface – OMIS

The monitoring interface is crucial for our system, since it defines each measurement and enables to control it. This interface should make it possible to easily define various monitoring activities and at the same time should be extensible. The On-line Monitoring Interface Specification (OMIS) [11] suits these requirements well.

The key concepts in OMIS are *objects* and *services*. Objects identify the entities in the target system such as processes, nodes, sites, etc., while services define the monitoring functionality.

Each object has an identifier, a so called *token*, for example `n_1` (node 1), `p_11` (process 11), `app_22` (application 22). Monitoring services are divided into three classes: information services to obtain information about objects (e.g. `node_get_info`), manipulation services to manipulate objects (e.g. `thread_stop`), and event services to detect events (e.g. `thread_creates_proc`).

The services can be combined into monitoring service requests. There are two types of requests: unconditional and conditional ones.

- The unconditional service requests are composed of one or more information and/or manipulation services (referred to as actions). They result in an immediate action in the target system, for example *stop process p*; in OMIS syntax: `: thread_stop([p])`.
- The conditional service requests (CSR) are composed of an event service and a list of actions. Their semantics is as follows: whenever the event occurs, execute the actions, for example *whenever process p starts a new process, get information about that process*; in OMIS syntax:
  `thread_creates_proc([p]) : proc_get_info([$newproc], 0x1)`.

The concepts of OMIS fit well in the GMA [8], the monitoring architecture defined by Global Grid Forum. This is illustrated in Tab. 1, in which we can see the corresponding features of OMIS and GMA. The only difference is that in OMIS there is no notification interaction; this is because there is not default monitoring in OMIS and every monitoring action must be subscribed.

GMA	**OMIS**
producer	low level monitor or any other component that sends data
consumer	a tool or any other component that receives data
publish/subscribe interaction	Conditional Service Request
query/response interaction	Unconditional Service Request
notification interaction	—

Tab. 1: OMIS and GMA

4 Target Interface

The target interface specifies the low-level methods for performing monitoring actions specified by a tool. The implementation of OMIS services can use every method that is allowed by operating system to get a required measurment.

Some part of services are performed using the `/proc` file system and the `ptrace` system call. This enables, for instance, to stop process or to do some other manipulations. Another method to get the information is to make use of the Application Module. The communication between the Local Monitor and the Application Module is through a memory segment shared between these procceses. There is also a possibility to spawn external programs to do some measurments; a good example of this is a micro benchmark that measures an actual point-to-point network link latency.

5 Request Distribution

The request distribution mechanism in the OCM-G is illustrated with an example which shows the distribution of a sample request (see Fig. 3). The request is `thread_stop` which is used to stop processes. The argument given to this services is a list of processes to be stopped, in this case it is `a_1` which identifies the whole application and is expanded by the Service Manager to the list of all the processes of the application. Since the four processes of the application are distributed across two different sites and two nodes in the local site, the SM splits the request into subrequests and forwards each to the appropriate LMs and to SM on the remote site, which forwards the message to the appropriate LM (LM3). Every LM executes the request – it stops the processes listed in the request.

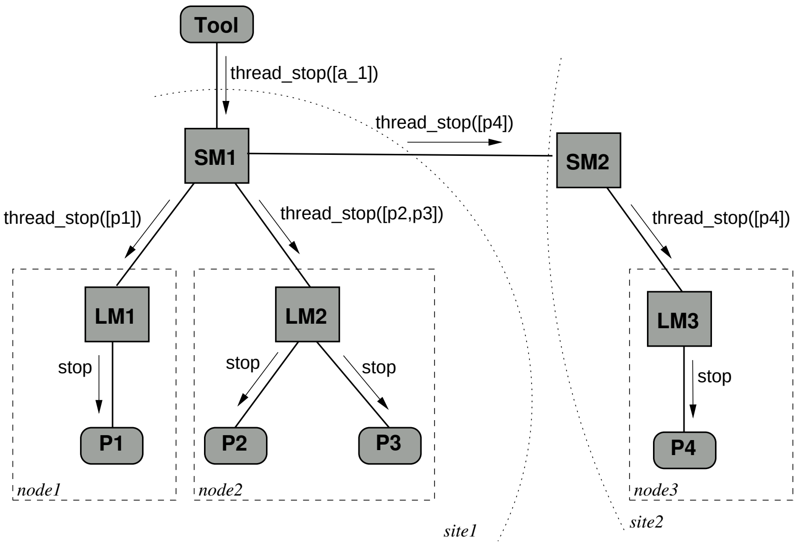

Fig. 3: Distribution of an OMIS request

To enable operation of the described algorithm, the Service Managers are required to have knowledge about the localization of application processes. However, if each Service Manager had to know everything about each application, it could be dangerous for the scalability of the whole system. Fortunately, it is not necessary. In fact, the mechanism is as follows. All the SMs which are involved in one application form a kind of Virtual Monitoring System (VMS) for that application. One of these SMs is designated for the role of a MainSM for that VMS. The MainSM is supposed to be the information service for the application, i.e. a) its location has to be well-known, b) it must be immediately notified about any changes in the application's state (process migration, creation, etc.), c) it has to know about all other SMs in the VMS so that it can notify them about important events. The only problematic issue is how to make the MainSM well known. The answer is that an external information service will be used for that, in which a mapping 'application-MainSM location' will be kept.

6 Managing Conditional Service Requests

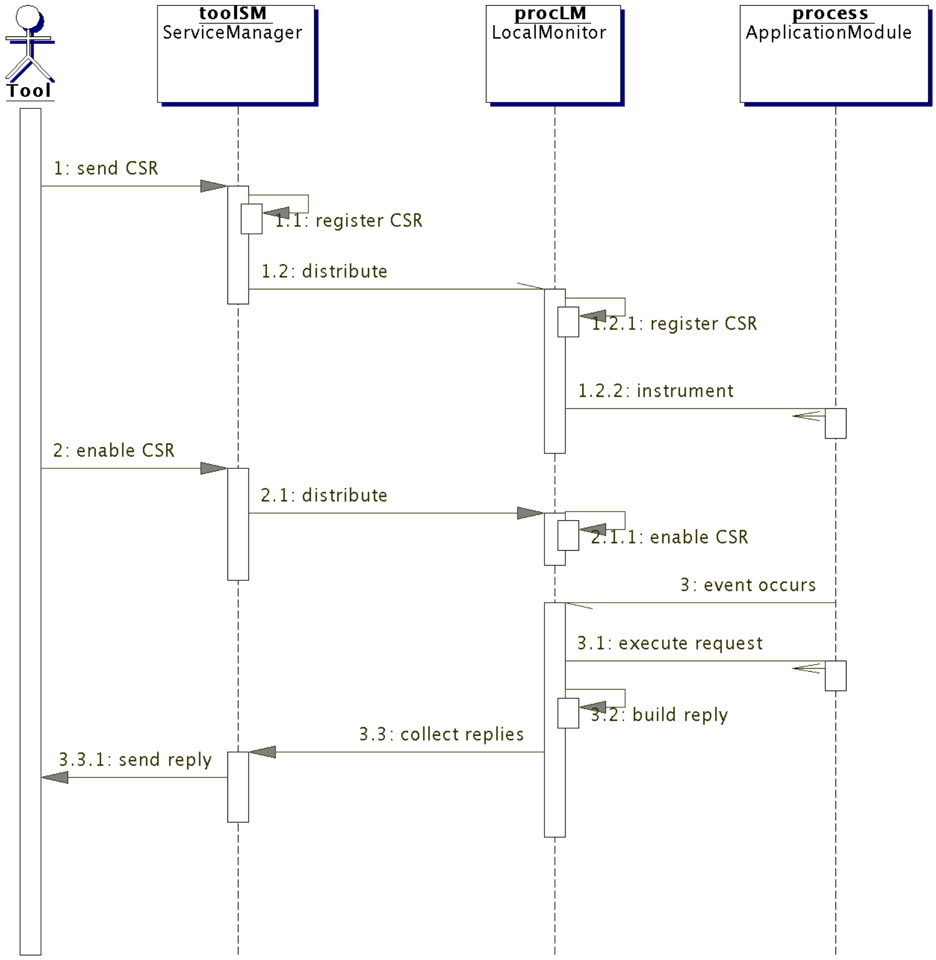

Fig. 4: Performing a Conditional Request

The majority of monitoring actions are event driven. These actions are set-up using OMIS conditional service requests. Fig. 4 shows a sequence diagram in UML convention with an example scenario of issuing a conditional request.

After the request has been submitted (message 1), it is registered in the SM (1.1), distributed to LMs (1.2) and again registered in LMs (1.2.1). The registration is necessary because, e.g., the SM has to keep the information from which tool the request has been submitted, while in LMs the actions have to be set up, etc. In addition, some instrumentation may be set up (or activated, 1.2.2) in the application context to enable tracking some kinds of events, e.g., function calls. Next, the tool can send a request to enable the CSR (message 2). After distribution (2.1) and enabling it in LMs (2.1.1), the CSR is active. This means, when, at some time, the event occurs in an application process it is reported to the LM (message 3), which executes the corresponding actions (3.1). Next, a reply collection follows as in case of unconditional requests (3.3, 3.3.1).

Performing a CSR according to the design presented above results in a relatively small monitoring overhead, because events are catched and the associated actions are executed only if this has been explicitly requested by a tool.

7 Start-up of the Monitoring Environment

The start-up procedure of the monitoring environment involves three elements: the application, a tool, and the OCM-G itself. The initial assumptions and requirements are the following:

- the mechanism to submit an application to the Grid should not change,
- a user should be able to start a tool at any time and begin monitoring, even if the application is already running.

7.1 OCM-G Start-up

To properly start-up the monitoring system, first its components (LMs and SMs) must be started up, next, they must connect to assemble a single monitoring infrastructure. In the OCM-G, each component starts separately, i.e., is not spawned by some other one.

The Service Managers, being the permanent entities, are supposed to start with the Grid middleware. The location of the SM of a given site is supposed to be well-known on each host of this site. Therefore, it must be accessible in some kind of information service. In the simplest case, where the site incorporates a shared file system, the location of the SM can be stored in a well-known file.

The Local Monitors are supposed to run only on those hosts, on which there are some processes to be monitored. Thus, the Local Monitors are simply forked off by the application processes which are started by the Grid job managers. It must be stressed that this method has a few important advantages. First, it is the easiest and perhaps the only possible way to ensure that Local Monitors are started exactly on the same nodes and exactly at the same time as the application processes start. Second, this method is a portable one: no interaction with Grid services is necessary to know the location of the application's processes (see also next subsection). Once a LM is started, it refers to the external information service to learn about the location of its Service Manager, an registers in it.

7.2 Application Start-up

One of the principal requirements is that the application is started in a normal way. In addition, it should not be required to add special features to the Grid job submision systems in order to cooperate with OCM-G and, on other hand, OCM-G should still be able to monitor the application from its very beginning.

However, to enable monitoring, the application has to undergo some preparations. First, special monitoring libraries have to be linked to the application, which provide the functions for registering the processes in the monitoring system, etc. Second, instrumented versions of libraries have to be linked. This is necessary to enable detection of such events as calls to communication functions. Finally, the OCM-G requires some special command-line parameters to be passed to the application. The most important parameter is the name of the application, which is arbitrarily chosen by the user. This name, combined with the user's Grid ID, forms the unique application identifier. Only this identifier enables that a collection of processes can actually be recognized as being part of one application. Another parameter specifies the behaviour of the application's processes after the start-up – should the processes continue running or be stopped waiting for a tool to continue them. The possibility to stop the processes allows to monitor them from the very beginning – the tool issues a continue request after all the measurements have been defined.

Immediately after a process is started, it invokes the ocmg_register() function, which does two things: 1. it checks whether the LM exists, and forks it off, if not, and 2. it registers in the LM. Upon registration, the process sends all important information (e.g., its pid) to the LM, which forwards it to its SM, which in turn sends it to the MainSM. In this way, the process becomes known in the monitoring system.

Note that upon starting the application, the MainSM must be chosen in some way. At present we require that the user provides the location of the MainSM in a command line parameter, though a more convenient solution may be invented in the future. The first OCM-G prototype does not yet handle multiple sites, so at the moment the problem is irrelevant, anyway.

7.3 Tool Start-up

When the user starts a tool, he must provide the name of the application he wants to monitor. The name must be the same that was passed as a command line parameter when the application has been submitted.

First, the tool connects to the monitoring system. The first OCM-G prototype handles only one site, so the tool just connects to the single existing SM, however in later prototypes the situation is more complicated. The problem is that to get the information about the application, its MainSM should be contacted, while the tool may be even started from a site, on which no processes of this appilcations are running. The possible solution is as follows: the tool still can connect to any SM it wants (probably the one of the local site). Then, when the tool issues the request to attach to the application, the SM to which the tool is connected must find the MainSM for the application and forward the request to it. This is possble, since the information about the MainSM is available via an external information service, as explained in Section 5.

When the tool is connected to the monitoring system, and it has the name of the application, it issues an `app_attach2` request, which attaches to the application and returns its token, which is needed for further requests. Next, the tool requests the `app_get_proclist` service, via which it gets the tokens and locations of all processes of the application. Once this information has been received, the user can define the measurements. If the application processes were stopped, the tool will issue a `thread_continue` request upon activation of the measurements.

8 Status and Future Work

Currently the first prototype of the OCM-G is in a test phase. The implementation of this prototype was based on a previously existing OMIS-compliant monitoring system for clusters, the OCM[15]. The first prototype works on one site only and supports only one user. It provides all services defined by the OMIS 2.0 document, except for those which were undesirable in the Grid (e.g., for creation of a new process). Some of the new Grid-specific services were also implemented. The first prototype of the OCM-G implements the new Grid-enabled start-up scheme. The security mechanisms are not yet incorporated to this version.

The plans for the future prototype, on the way to the final release, are as follows:

- support for multiple sites,
- handling of dynamically changing applications,
- support for multiple users,
- provide security based on GSI,
- implement all the new Grid-specific services.

Most of the features above are already designed, but they still wait for the next implementation phase. The authors are convinced, that after this phase is completed, the OCM-G will be a sophisticated monitoring system, that can be the base of many types of tools supporting application development in the Grid.

Acknowledgments The authors would like to thank Dr. Włodzimierz Funika for his comments and help.

References

1. Balaton, Z., Kacsuk, P., Podhorszki, N., and Vajda, F.: Comparison of Representative Grid Monitoring Tools. http://hepunx.rl.ac.uk/edg/wp3/meetings/budapest-jan01/NP%20Grid%20Tools.pdf

2. Baliś, B., Bubak, M., Funika, W., Szepieniec, T., and Wismuüller, R.: An Infrastructure for Grid Application Monitoring In: Kranzlmueller, D., Kacsuk, P., Dongarra, J., Volker, J. (Eds.): Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proc. 9th European PVM/MPI Users' Group Meeting, Linz, Austria, September/October 2002, LNCS 2474, pp. 41-49, 2002
3. Bubak, M., Funika, W., Baliś, B., and Wismüller, R.: On-line OCM-based Tool Support for Parallel Applications. In: Yuen Chung Kwong (ed.): Annual Review of Scalable Computing, 3, Chapter 3, 2001, Singapore
4. Bubak, M., Funika, W., and Wismüller, R.: The CrossGrid Performance Analysis Tool for Interactive Grid Applications. In: Kranzlmueller, D., Kacsuk, P., Dongarra, J., Volker, J. (Eds.): Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proc. 9th European PVM/MPI Users' Group Meeting, Linz, Austria, September/October 2002, LNCS 2474, pp.50-60, 2002
5. Byrom, R., et al.: R-GMA: A Relational Grid Information and Monitoring System. This volume.
6. CrossGrid Project: http://www.eu-crossgrid.org
7. Foster, I., Kesselman, C. (eds.): The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1999
8. Grid Monitoring Architecture. http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-2.pdf
9. Gunter, D. et al.: NetLogger: A Toolkit for Distributed System Performance Analysis. Proc. IEEE Mascots 2000 Conference, Lawrence Berkeley Natl. Lab., Report LBNL-46269
10. Heartbeat Monitor Specification. http://www.globus.org/hbm/heartbeat_spec.html
11. Ludwig, T., Wismüller, R., Sunderam, V., and Bode, A.: OMIS – On-line Monitoring Interface Specification (Version 2.0). Shaker Verlag, Aachen, vol. 9, LRR-TUM Research Report Series, (1997) http://wwwbode.in.tum.de/~omis
12. Podhorski, N., Kacsuk, P.: Design and Implementation of a Distributed Monitor for Semi-on-line Monitoring of VisualMP Applications. Proc. DAPSYS 2000, Balatonfured, Hungary, 23-32, 2000
13. Vetter, J.S., and Reed, D.A.: Real-time Monitoring, Adaptive Control and Interactive Steering of Computational Grids. The International Journal of High Performance Computing Applications **14** 357-366 (2000)
14. Waheed, A., et al.: An Infrastructure for Monitoring and Management in Computational Grids. Proc. 5th Workshop on Languages, Compilers and Run-time Systems for Scalable Computers, (March 2000)
15. Wismüller, R., Trinitis, J., and Ludwig T.: OCM - A Monitoring System for Interoperable Tools. In: Proceedings of the 2nd SIGMETRICS Symposium on Parallel and Distributed Tools SPDT'98, Welches, OR, USA, August 1998
16. Wolski, R., Spring, N., and Hayes, J. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Future Generation Computer Systems **15** 757-768 (1999)