Integrated Monitoring Framework for Grid Infrastructure and Applications*

Bartosz Baliś[1], Marian Bubak[1,2], Jakub Dziwisz[1],
Hong-Linh Truong[3], Thomas Fahringer[3]

[1]*Institute of Computer Science, AGH University of Science and Technology*
Email: balis@agh.edu.pl
[2]*Academic Computer Centre - CYFRONET, Krakow*
Email: bubak@uci.agh.edu.pl
[3]*Institute for Computer Science, University of Innsbruck*
Email: {truong,tf}@dps.uibk.ac.at

Abstract: This paper describes a monitoring framework called GEMINI (Generic Monitoring Infrastructure), developed as part of the EU-IST Project K-WfGrid, intended to collect, store and deliver diverse monitoring information about a variety of resources including Grid infrastructure and applications. GEMINI aims at providing a generic sensor that can be used to capture various types of monitoring data and a novel decentralized network of Web Service-based monitoring services for monitoring the execution of workflows of Grid services. GEMINI exposes interface for instrumentation of monitored applications that is essential to enable acquisition of information about running applications. The metrics are described in a form of metadata OWL-based ontological description and the communication between clients, monitors and sensors is based on standard XML-based protocols.

1. Introduction

Monitoring services are important part of many distributed systems and are essential in Grid environments. Entities monitored in a Grid environment encompass infrastructure elements, applications, middleware, and others. Here are some examples in which monitoring information is important or essential: (1) brokering services need up-to-date information about the availability of resources to enable correct resource allocation and job submission, (2) dynamic load balancers may use information about the resource usage to decide process migration, (3) data access optimization services require information about bandwidth and latency of network paths to choose proper file replica [4], and (4) application performance analysis tools [6] need to monitor application's execution to visualize its performance on graphical charts [13].

The main objective of the K-WfGrid EU IST Project entitled "Knowledge based Workflow system for Grid Applications" [10] is to provide the knowledge-based support for workflow construction and execution [3]. That is, K-WfGrid will semi-automatically compose a workflow of Grid services. In do so, besides capturing knowledge about functional information of workflows, K-WfGrid has to monitor the performance of the Grid infrastructure and workflow applications, analyze the resulting monitoring information, and capture the knowledge from monitoring information. Then, K-WfGrid

* This work is supported by EU-IST Project K-WfGrid – Knowledge Based Workflow System for Grid Applications, IST-2002-511385, http://www.kwfGrid.net.

uses joining knowledge gathered from all participating users, from functional information and monitoring information to efficiently construct workflows for new Grid applications.

In this paper we describe how GEMINI monitoring framework is designed in to support knowledge-based Grid service-based workflow construction and execution. On the one hand, the monitoring service has to provide detailed information associated with multiple levels of abstraction including workflow, workflow region, workflow activity, invoked application, etc. One the other hand, monitoring data has to be presented in a well-defined form so that knowledge can be easily extracted from the data. The paper focuses on architecture, sensor deployment, and metrics. In K-WfGrid GEMINI is responsible for collection and delivery of any monitoring information, whether static or dynamic, required by other components of the system, which are, among others, performance analysis services, scheduling system, and knowledge-building modules.

2. Related work

The process of monitoring involves several stages, just to name a few important stages:

1. *Data collection*. This is the actual extraction or evaluation of a concrete piece of data; this process frequently requires *instrumentation* of the monitored entity. For example, to obtain current the CPU load of machines, some monitoring agents may exist on target machines and use *ps* command for doing this. Or, to evaluate time taken to invoke a web-service method, it may be needed to instrument the web-service container by inserting additional code before and after the code invoking the method.
2. *Data storage*. In some cases the collected information may be stored in a repository for a later reference, either as a statistic or raw-trace data. For example, historic information about CPU usage, network load or system availability may prove useful for prediction or diagnosis purposes. Static or semi-static information such as resources connected to the Grid with their parameters (OS version, available memory, etc.) obviously needs to be stored in a registry. Sometimes the data is stored simply in a file, for example a trace of an application's execution in case of off-line application monitoring.
3. *Data delivery*. After the monitoring data has been collected, it must be delivered to clients who request it. The data can be accessed on-line, i.e., when a client requests interesting data, the requested data is collected dynamically and returned to the client. Other possibility is to access data off-line, e.g., from files or database.

Recently two new standards for Web Service Management have emerged. First of them, Web Services Distributed Management (WSDM) is a specification developed by an OASIS TC. It is an integration layer that resides between managers and the different protocols used to control resources. It consists of two specifications: (1) Management Using Web Services (MUWS) addressing usage of Web services to facilitate interactions between managed resources and management applications across numerous vendors, platforms, technologies, and topologies, (2) Management of Web Services (MOWS) addressing the specific requirements for managing Web services like any other IT resource.

The second specification, WS-Management, published by AMD, Dell, Intel, Microsoft and Sun, also deals with managing devices and computers using Web Services. It defines a way to use the existing set of core Web services specifications and it covers only a small part of what WSDM is attempting to address, i.e. manageable resource identification and communication. By that means it is more lightweight than WSDM.

Both these standards offer no real support for resource location in highly dynamic and distributed over different administrative domains environment as one GEMINI is dealing with. The other problem is that GEMINI aims to monitor resources. That is why minimal efficiency overheads (both network and computational) are so important. That is also the reason for using specific (but open source) distributed technologies (eg. Internet Communication Engine, ICE). At the same time we are taking interoperability into consideration by providing SDKs for different platforms.

However we are considering support for Web Service Management and Notification in further versions of GEMINI. Though usage of such connectivity layer is not optimal, in some cases would be the only available for system administrators. But due to limitations pointed out earlier this layer would be intended to use only if there are no other options.

Current monitoring systems are usually specialized, for example focus only on application or infrastructure monitoring (see [6][17] for a greater detail). Those systems are tailored to collect and deliver a particular type of data which is reflected in their internal design and interface exposed to clients to obtain the monitoring data. Instead of monitoring applications or infrastructure separately, we proposed a unified approach to the performance monitoring and analysis for the Grid [16]. That is, the performance of both applications and infrastructure should be monitored and analyzed in an integrated system [16]. To integrate a variety of monitoring data of both applications and infrastructure in the Grid, it necessitates having integrated and generic frameworks for measuring and collecting these diverse monitoring data. However, until now little effort has been spent in the development of such frameworks.

Rare efforts to provide a *generic* monitoring infrastructure are R-GMA [8] and Mercury Monitor [11] [12]. R-GMA employs an approach based on relational data model. Though, R-GMA is not a distributed relational database management system, it makes the information appear as a large relational database in which data may be published and requested via SQL-like queries. Mercury is a monitoring system which provides monitoring data as metrics and also supports steering. Mercury provides sensors for monitoring of application and resources. Its modular design enables easy extension with new metrics. However, in both R-GMA and Mercury monitoring data is stored in relational database. This results in slow access to monitoring data.

On the one hand, these existing projects do not concentrate on monitoring Grid service-based workflows which are inter-organizational and service-oriented. We note that these existing projects are not focusing on knowledge-based description for monitoring data that they provide. In K-WfGrid, metrics have to be well-described for extracting knowledge from monitoring data. Our goal is to design and develop a generic Monitoring Framework, called GEMINI, which supports collection, delivery, and storage of arbitrary types of data. The monitoring data is described in the form of metrics. Data is produced by *sensors*. We provide a generic sensor library which may be used by any entity to easily connect to the monitoring framework and produce data. Data may be obtained by clients via *query* or *subscribe* requests. Unique capabilities of our approach are as follows:

1. Sensors are configurable, for example, they may be activated or deactivated at any time. Sensors can be event-driven or demand-driven and with/without rule-based monitoring capabilities [15].
2. The monitoring data is not only stored in a fast repository but also delivered directly to clients on the fly.

3. Metrics are described in a form of ontological description. This supports knowledge extraction.
4. The Monitoring Framework is distributed and decentralized, based on peer-to-peer model [19], to support scalability, availability and fault-tolerance.
5. The monitoring services are available as web services to enable interoperability while the actual communication between clients and sensors is based on a low-level, but fast, mechanism to ensure efficiency.

2. GEMINI Monitoring Framework

Figure 1 shows the architecture of the GEMINI and related components. The monitoring framework itself consists of (1) Monitoring and Instrumentation Service (MIS) which contains Event Infrastructure and Data Repository, and (2) a set of diverse sensors.

The Monitoring and Instrumentation Service forms in fact a peer network of interconnected Monitors (shown in Figure 2, described below). Diverse sensors connect to the MIS network via any Monitor and register the type(s) of data they provide. The data types are expressed as metrics, while each metric has a well-defined name and specification. Each Monitor in the MIS network has a data repository in which the monitoring data can be temporarily stored. The MIS network also plays a role of Event Infrastructure in which events can be published and subscribed and through which the events are transported from producers to consumers. Grid Organizational Memory (GOM) [9] is an external service which fulfils the role of a registry in which web services exposed by the MIS network as well as metadata describing data provided by the monitoring framework are registered. Clients refer to GOM to find out what data is available and discover the monitoring and instrumentation services. Clients request monitoring data in two modes: (1) query – single request/response, and (2) subscribe – request for data stream over a certain time period.

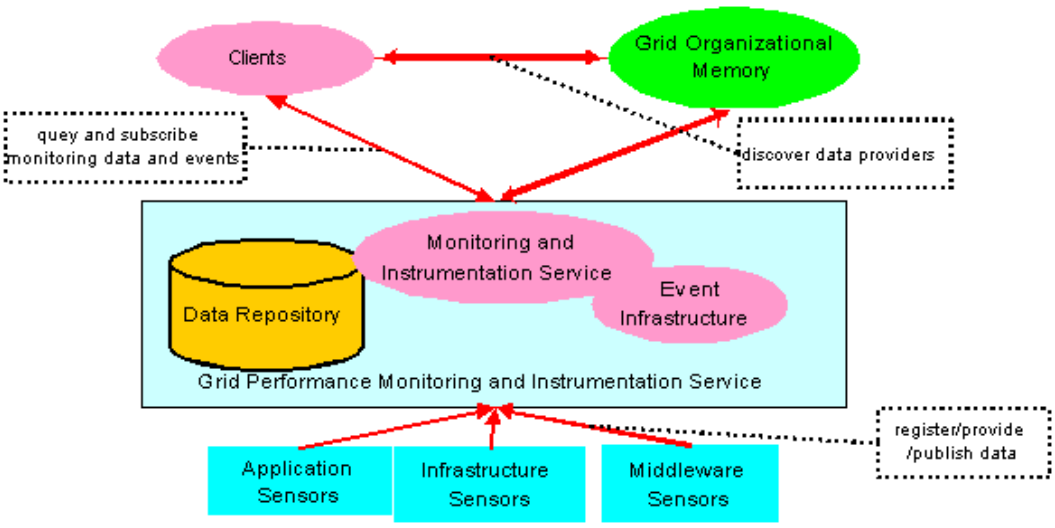

Figure 1: GEMINI: Generic Monitoring Infrastructure – overview architecture

Figure 2 presents the distributed architecture of the GEMINI Framework. The MIS network is composed of a distributed and decentralized set of Monitors. Monitors are organized hierarchically, thus there are upper-layer Domain Monitors ("D-Monitors") and lower-layer "Monitors". This organization into a super-peer topology increases the scalability of the system. A relatively low number of D-Monitors manage underlying Monitors. Thus the information needed for system management is shared only between D-Monitors.

Each D-Monitor exposes two separate web-service interfaces: Monitoring Interface and Instrumentation Interface. Clients use those interfaces for requesting monitoring data and also for issuing application instrumentation requests. The actual data transfer from sensor(s) to the client (not shown) is direct and based on a low-level communication mechanism for efficiency reasons. Some problems related to communication between clients and sensors are as follows:

1. Sometimes the direct communication may be difficult due to firewalls and the fact that sensor could frequently reside in private networks. In such cases it will be possible to route the transfer via the Monitors network or use relay servers.
2. In principle it should be possible for the client to issue aggregate requests in which the same metric is requested at once for multiple resources (e.g. CPU load on machines X, Y and Z). We handle those cases transparently, i.e. the client does not have to handle streams from multiple sensors and aggregate the data himself.

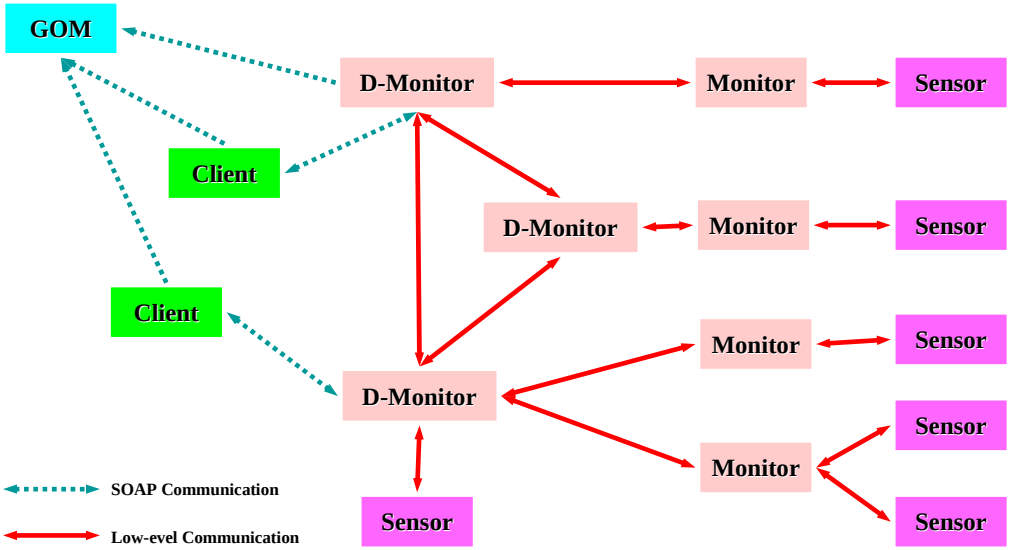

Figure 2: Distributed Architecture of GEMINI Monitoring Framework

3. Sensors

We provide a *generic sensor library* which can be incorporated into any entity to produce the monitoring data into the GEMINI framework. In order to publish the monitoring data, the following steps are necessary:

- Link the generic sensor library into the entity supposed to produce the information (an application, scheduling system, workflow execution engine, etc.)
- Specify a sensor description containing information of provided metrics and resources for which the metrics are supported (for example – metric *CPU usage* for machine X, Y and Z, *function call count/delay* for MPI_Xxxx, etc.) This description is expressed in an XML-based language, and besides the metrics and resources, it specifies the class/function names which implement the metrics.
- Implement the code which actually extracts the data (for example reads the current CPU usage using *ps* command).

It is important that while the user[†] has to implement the code to evaluate the monitoring data, the user does not need to care about anything else, specifically sensor initialization, communication between the sensor and monitoring framework, etc. Those activities are performed transparently via the generic sensor library. The user's code is incorporated automatically as a plug-in.

The sensor model in GEMINI is an extension of our previous sensor model for Grid monitoring and data integration [15]. Several types of sensors will be supported, for example, event-driven sensors (sensors deliver monitoring data upon an event), demand-driven sensors (sensors provide monitoring data upon a request) and with/without rule-based monitoring (sensors use rules to control their actions, analyze monitoring data and react with appropriate functions). The sensors will have customizable features. For example, sensors can customize the structure of monitoring data and buffering policy.

4. Metrics

The metrics provided by the monitoring framework is well classified. Metrics are associated with multiple levels of abstraction such as code region, invoked application, activity, etc., and supportive metrics are described by a metric ontology [14]. To help the client to query and subscribe metrics provided by the monitoring at runtime, information about every metric monitored is described in OWL (Web Ontology Language) [18]. This information will be published into GOM so that any client can locate the monitoring service that provides metrics of interest. For example, Figure 3 presents a draft ontology-based description for metrics. Other aspects will be investigated. The sensor (metric) name is specified by property *ofSensor*. The properties *validFrom* / *validTo* specify the duration in which the data is available. The property *isStoredIn* provides the service handle of the web service providing the monitoring data.

Figure 3: Description of OWL/RDF information of monitoring data

```
<MonitoringData rdf:ID="host.cpu.used">
  <validFrom>12345678</validFrom>
  <isStoredIn>
     http://bridge.vcpc.univie.ac.at:8765/ogsa/services/MonitoringService
  </isStoredIn>
  <ofResource>schareck.dps.uibk.ac.at</ofResource>
  <validTo>12345687</validTo>
  <ofSensor>host.cpu.used</ofSensor>
</MonitoringData>
```

Figure 4: Example of an OWL description of "host.cpu.used" metric (simplified)

[†] The user of the monitoring framework is not the end-user, but the developer of services, applications, middleware, etc.

Figure 4 presents an example of OWL description of metric "host.cpu.used" (simplified). This metric specifies the CPU usage of a machine; CPU usage includes detailed information about CPU idle, user and system time for every CPU of a machine.

Table 1 lists some application metrics. Detailed proposed application metrics can be found in [14]. Table 2 presents a few infrastructure metrics. Given a large number of infrastructure monitoring tools, (see [6]), we do not intend to develop a new infrastructure monitoring; only missing functions will be developed. Instead, we focus on the integration of existing infrastructure monitoring tools into our framework, making tool-specific metrics available and accessible through well-defined representations (e.g., XML and OWL-based ontology description) and interfaces (e.g., Web Services) and providing these metrics to the client.

Table 1: Performance metrics related to application monitoring at code region level

Category	Metric Name	Description
Execution time	ElapsedTime	The elapsed time of the code region.
	UserCPUTime	CPU time spent on user mode
	SystemCPUTime	CPU time spent on system mode
	CommTime	Communication time
Counter	NCalls	Number of executions of the code region
	NSubs	Number of executions of sub regions of the code region
	TotalTransSize	Size of total data transferred (send and receive)

Table 2: Static and dynamic infrastructure metrics

Category	Metric Name	Description
Static information	HostName	Machine name
	IPAddr	IP address
	OpSys	Operating System
	CPUType	CPU Type and information
	MaxMainMemSize	Maximum main memory size
	MaxDiskSize	Maximum disk size
Dynamic information	MemUsage	Current memory usage
	CPUUsage	Current CPU usage (user/system/ idle)
	Avail	Availability of a resource
	NetPathBandwidth	Current path bandwidth
	NetPathLatency	Network path latency
	NetPathAvail	Network path availability (site A can be reached from site B)

5. Summary and Status

We have presented a generic Monitoring Framework GEMINI which fulfils the role of a generic, efficient infrastructure for collection, storage and delivery of arbitrary monitoring information. The framework consists of a core network of distributed and scalable Monitors which are designed to allow new data sources to dynamically connect and deliver data. Generic sensor infrastructure was built for this purpose.

The GEMINI framework is developed as part of the K-Wf Grid project [10] in which it will be the supportive monitoring middleware for performance analysis of workflow applications and an information source to enable extraction of knowledge about the operation of the Grid environment.

The K-WfGrid Project will close the gap between monitoring and workflow construction – our Grid system will be able to learn how to perform tasks and subsequently

will employ this knowledge in workflow construction. The Project will combine together and extend current achievements in Web Services, Semantic Web, Grid services to enable easier composition, execution and monitoring of workflows for real-life applications. We expect that many environmental, industrial, state and municipal authorities will be the potential users of technologies developed within the Project.

At present we are about to release a first prototype of the described framework. It is being implemented in a Linux/GT4 (Globus Toolkit 4) [7] environment. We also work on adaptation of existing systems monitoring systems to work as sensor for the framework, namely Ganglia [5] for infrastructure metrics and OCM-G [1][2] for legacy MPI application monitoring.

References

[1] B. Balis, M. Bubak, W. Funika, R. Wismueller, M. Radecki, T. Szepieniec, T. Arodz, M. Kurdziel: *Grid Environment for On-line Application Monitoring and Performance Analysis*, Scientific Pogrammning, vol. 12, no. 4, 2004, pp. 239-251.

[2] B. Balis, M. Bubak, M. Radecki, T. Szepieniec, R. Wismueller: *Application Monitoring in CrossGrid and Other Grid Projects*. In: In Dikaiakos M., editor, Grid Computing. Proc. Second European Across Grids Conference, pages 212-219, Nicosia, Cyprus, January 2004, Springer.

[3] M. Bubak, T. Gubala, M. Kapalka, M. Malawski , K. Rycerz: *Workflow composer and service registry for grid applications*, Future Generation Computer Systems, vol. 21, no. 1, 2005, pp. 79-86.

[4] L. Dutka, R. Slota, D. Nikolow, J. Kitowski: *Optimization of Data Access for Grid Environment*, in: Rivera, F. F., et al. (Eds.), Proceedings of Grid Computing, First European Across Grids Conference, Santiago de Compostela, Spain, February 2003, Lecture Notes in Computer Science, no. 2970, Springer, 2004, pp. 93-102.

[5] Ganglia homepage: http://ganglia.sourceforge.net

[6] M. Gerndt, R. Wismueller, Z. Balaton, G. Gombas, P. Kacsuk, Z. Nemeth, N. Podhorecki, H.-L. Truong, T. Fahringer, M. Bubak, E. Laure, T. Margalef, *Performance Tools for the Grid: State of the Art and Future, APART White Paper*, Shaker Verlag, 2004, ISBN 3-8322-2413-0.

[7] The Globus Project homepage: http://www.globus.org

[8] R-GMA homepage: http://www.r-gma.org

[9] K. Krawczyk, R. Slota, M. Majewska, B. Kryza, J. Kitowski: *Grid Organization Memory for Knowledge Management for Grid Environment*, in: M. Bubak, M. Turala, K. Wiatr (Eds.), Proceedings of Cracow Grid Workshop, CGW'04, December 12-15, 2004, Cracow, Poland, ACC CYFRONET AGH, 2005.

[10] K-WfGrid project homepage: http://www.kwfGrid.net

[11] The Mercury Grid Monitoring System homepage: http://www.lpds.sztaki.hu/mercury

[12] N. Podhorszki, Z. Balaton, G. Gombas. *Monitoring Message-Passing Parallel Applications in the Grid with GRM and Mercury Monitor*. In: In: Dikaiakos M., editor, Grid Computing. Proc. Second European Across Grids Conference, Nicosia, Cyprus, January 2004, Springer.

[13] R. Wismueller, M. Bubak, W. Funika, B. Balis, A Performance Analysis Tool for Interactive Applications on the Grid, Intl. Journal of High Performance Computing Applications, vol. 18, no. 3, pp. 305-316, 2004.

[14] H.-L. Truong, T. Fahringer, F. Nerieri, S. Dustdar: *Performance Metrics and Ontology for Describing Performance Data of Grid Workflows*, 1st International Workshop on Grid Performability, held in conjunction with IEEE International Symposium on Cluster Computing and Grid 2005 (CCGrid2005), IEEE Computer Society Press, Cardiff, UK, 9 - 12 May 2005.

[15] H.-L. Truong, T. Fahringer: *Self-Managing Sensor-based Middleware for Grid Monitoring and Performance Data Integration*, 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), IEEE Computer Society Press, Denver, Colorado, USA, April 4-8, 2005.

[16] H.-L. Truong, T. Fahringer: *SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid*, Scientific Programming, 12(4):225-237, IOS Press, 2004.

[17] S. Zanikolas, R. Sakellariou: *A Taxonomy of Grid Monitoring Systems*, Future Generation Computing Systems,vol. 21, no. 1, pp 163-188, January, 2005.

[18] OWL Web Ontology Language Reference, http://www.w3.org/TR/owl-ref/

[19] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, Z. Xu, *"Peer-To-Peer Computing"*, Technical report, HPL-2002-57, HP Labs, March 2002.