Towards a Framework for Monitoring and Analyzing QoS Metrics of Grid Services *

Hong-Linh Truong, Robert Samborski, Thomas Fahringer
Institute of Computer Science, University of Innsbruck
Email: {truong, robert, tf}@dps.uibk.ac.at

Abstract

QoS (Quality of Service) parameters play a key role in selecting Grid resources and optimizing resources usage efficiently. Although many works have focused on using QoS metrics, surprisingly few tools support the monitoring and analysis of QoS metrics of Grid services. This paper presents a novel framework which supports the monitoring and analysis of QoS metrics in the Grid. Our approach is that, firstly, we develop a classification of important QoS metrics for Grid services that should be monitored and analyzed. Secondly, sensors are developed to monitor QoS of disparate Grid services by using a peer-to-peer Grid monitoring middleware. The dependencies among Grid services are modeled. Based on that, several techniques are used to analyze QoS metrics of dependent Grid services.

1 Introduction

Grid computing opens opportunities for utilizing various resources from different organizations. Grid resources provided by these organizations are complement but also redundant. With the Grid, the user is able not only to harness a powerful pool of resources but also to choose the best resources, suitable for his/her tasks, from various similar resources. In order to use these resources efficiently, users, for example, assisted by the resource management system and the scheduler, must be able to select the best resources for their tasks, based on quality aspects of resources. Not surprisingly, utilizing QoS parameters in the Grid has attracted a lot of attention because QoS parameters play a key role in selecting resources efficiently and in negotiating service level agreements (SLAs) between clients and services [25, 12, 37, 9, 24].

However, most of existing works on QoS focus on specifying and modeling QoS of Grid services and workflows [15, 26, 14] or on using QoS in service compositions, service negotiations, resource management systems, service discoveries, and schedulers [27, 20, 24, 13, 28]. Surprisingly, it is difficult to find out which techniques a framework uses to monitor and analyze QoS metrics and to provide these metrics to its clients. Moreover, there are few works on monitoring and analyzing QoS of Grid resources that consider the dependencies among these resources. In fact, we argued that monitoring techniques for QoS attributes in the Grid have been neglected and there is a lack of QoS monitoring and analysis tools for the Grid. This paper presents a framework for monitoring and analyzing QoS metrics of Grid services. We introduce a novel classification of QoS metrics and describe in detail how the framework can monitor QoS metrics, not only for individual Grid services but also for dependent Grid services by taking into account complex dependencies among Grid services. Interdependent services in the Grid are presented visually as a graph. Based on that, QoS metrics can be analyzed during runtime. In this paper, we also present a few experiments demonstrating the preliminary results of our work.

The rest of this paper is organized as follows: Section 2 analyzes related work and discusses about the motivation. Section 3 presents our classification of selected QoS metrics. Section 4 depicts the framework architecture. Detailed monitoring and analysis techniques are presented in Section 5. We present experiments in Section 6. Section 7 summarizes the paper and outlines the future work.

2 Related Work and Motivation

QoS has been studied for a long time, resulting in hundred QoS-related papers in different domains in literature. QoS parameters are associated with various aspects, normally considered as *non-functional* parameters, such as performance, dependability (including security) and cost. In the context of our research, we focus on monitoring QoS metrics of Grid services.

A taxonomy of QoS specifications given in [31] is a widely cited source for discussing QoS parameters. That

*This work is partially funded by the European Union through the IST-2002-511385 K-WfGrid and IST-034601 Edutain@Grid projects.

taxonomy classifies QoS attributes into metrics and policies. Metrics include security (confidentiality and integrity), performance (timeliness, precision, accuracy and combinations) and relative importance; policies include management and levels of service.

[28] presents a Web services (WS) discovery model that utilizes various QoS metrics like performance, reliability, integrity, accessibility, availability, capacity, scalability, security, etc. [8] and [32] also discuss similar QoS attributes for WS. In [6], requirements and approaches for QoS for WS are discussed. While the above-mentioned works discuss possible QoS metrics and approaches for WS, they do not explain how to monitor such QoS attributes.

In [21], QoS attributes named priorities, versions, deadline and security are discussed. Many discussed attributes are similar or related to those in [31]. For example, versions are related to precision and accuracy [21]. Patel et al. classify QoS parameters into general, Internet service specific and task specific [27]; main QoS parameters are latency, reliability, availability, security, accessibility and regulatory. Performance-related metrics are well understood. Most performance metrics are time- and ratio-based. Previously, we have introduced an ontology describing various performance metrics of Grid workflows [34]. A taxonomy of security services is presented in [19] whereas security-related requirements are discussed in [16].

Several papers discuss about using QoS for scheduling [9] and composing workflows and WS [22, 20, 13]. For example, [20] uses execution time, cost, encryption level, throughput and uptime probability for composing services. There are some QoS ontologies [15, 26] for service-based applications. These ontologies describe QoS vocabularies and metrics, and their relationships, rather than present which QoS metrics are suitable and how to monitor them.

We observed that while many QoS attributes are discussed for WS and Grid applications/workflows, existing techniques for measuring and monitoring QoS attributes are limited and inadequate. Using extra service proxies/wrappers and instrumented clients and services is also employed for monitoring QoS attributes of WS [29]. However, this method cannot be applied for different types of services developed in the Grid. Recently, WSDM (Web Services Distributed Management) [36] standard allows us to efficiently monitor and manage QoS of Grid services, however, it has not been commonly adopted yet. Many frameworks use QoS attributes, but are just based on simulation without the support of a real QoS monitoring tool.

Motivated by the lack of frameworks for monitoring and analyzing QoS attributes of Grid services, our goal is to deal with three important issues named monitoring, analysis and management of QoS attributes. As discussed above, we observed that several different QoS metrics are proposed in different works. However, the QoS taxonomy in [31] is not suitable for Grid resources since it has not considered various quality aspects of Grid computing. For example, as many Grid services, offered by different organizations, compete with each other, it turns out that QoS metrics specifying virtual organizations, Grid-specific security levels and standards are important. QoS metrics proposed in existing frameworks do not fully cover quality aspects of Grid services if we consider them individually. Therefore, a QoS classification for Grid resources that combines existing QoS metrics with new Grid-specific QoS metrics is needed.

Moreover, currently, manageability for Grid services has not been paid enough attention. However, due to the complexity of the Grid, managing Grid services is a key issue. Applying autonomic computing concepts to the management of Grid services is currently being increased. As a result, exploiting features of autonomic computing for monitoring and analysis of QoS attributes should be investigated. Grid services are diverse, thus any single method cannot be used for monitoring various types of Grid services. Therefore, we have to apply different measurement methods to different services and to unify these methods into a single framework. Because the Grid introduces complex interactions among various services, any QoS monitoring and analysis framework has to consider dependencies among Grid services and to support the monitoring and analysis of QoS based on these dependencies.

In this paper we discuss our first step to tackle the above-mentioned issues. We focus on determining QoS metrics for Grid services that should be monitored and on how to monitor and analyze them.

3 Classifying and Providing QoS Metrics for Grid Services

In our work, a Grid service is understood as a computational resource or a network path or a middleware service or a Grid application. In the Grid, services are typically distributed across different Grid sites. We use the term *monitored resource* to indicate a Grid service being monitored. In our study, we concentrate on supporting Grid applications which are based on *WS/WSRF* (Web Services Resource Framework) services and *workflows of Web/WSRF services*.

3.1 Classifying Selected QoS Metrics

As discussed in the previous section, we focus on QoS metrics of Grid services that can be monitored and measured. Figure 1 presents the classification of our selected QoS metrics. The classification is based on the taxonomy of QoS specifications in [31], the dependability taxonomy in [11], various metrics from existing works discussed in Section 2, and our previous work on performance metrics

for workflows [34]. At the top level, QoS metrics are classified into *Performance, Dependability, Configuration, Cost* and *CustomMetric*.

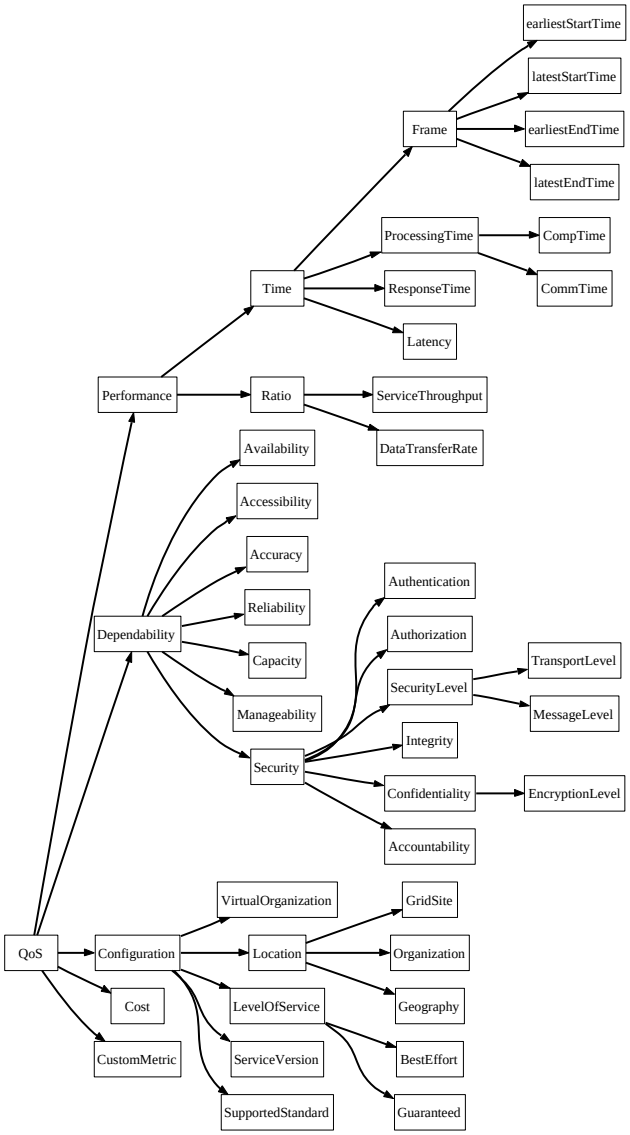

Figure 1. Classification of QoS metrics

Performance indicates how well a service performs. Performance is divided into *Time* and *Ratio*. The *Time* subclass includes various metrics such as *ResponseTime* (also called wall-clock time or elapsed time), *ProcessingTime* (defined as time spent on processing a request, for example to perform computation (*CompTime*) and communication (*CommTime*)), and frame-based time such as *earliestStartTime* (defined as the earliest start time), *latestEndTime* (defined as the latest end time), etc. The *Ratio* subclass indicates performance metrics which are computed based on ratio, for example, *ServiceThroughput* (ratio of requests served to a given time period), *DataTransferRate* (defined as data transfer rate), etc. The *Time* and the *Ratio* classes basically include well-known time metrics in literature. Many other performance metrics in [34] are in this category.

The *Dependability* subclass is based on dependability attributes introduced in [11]. However, different from [11], our dependability includes *Availability, Accessibility, Accuracy, Reliability, Capacity, Manageability* and *Security*. *Availability* defines whether a resource is ready for immediate use [8]. Availability is normally defined by $Availability = \frac{UpTime}{PeriodOfTime}$ where $UpTime$ is the fraction of time in the period $PeriodOfTime$ in which the service has been up. *Accessibility* defines whether a service is capable of serving requests [28, 32]. Note that while many services are ready for use, they might not be accessible to specific clients. In many cases a service is available for a client but another client could not access the resource due to some problems with the connection between the client and the service or because the service already reached its threshold. From client's perspective, it is interesting to distinguish the available status from the accessible status. *Accuracy* is defined as the ratio of the number of correct results to the total number of results. To better understand the relationship among *Availability, Accessibility* and *Accuracy*, we consider the following chain: $Availability \rightarrow Accessibility \rightarrow Accuracy$. In order to use a service, firstly the requester checks whether the service is available. Once it is available, the requester can send a request; if the request can be instantiated then the service is accessible. And then, if the requester gets the result, the result can be compared with the expected one to see how accurate the service is. *Reliability* is well defined in literature as the ability of maintaining service operations without failure and characterized by various parameters such as number of failures, mean time between failures (MTBF), and mean time to recovery (MTTR) [11]. *Capacity* is defined as the maximum simultaneous requests which can be supported with guaranteed performance [6, 28]. *Manageability* indicates how good a service can be self-managed, based on autonomic computing. We define *Manageability* as the ratio of successful self-recoveries to the total number of failures. *Security* indicates QoS parameters specifying security level in Grid services. [11] does not put security metrics into a separate subclass, although many security-related metrics are in dependability category, because security is normally viewed as a combination of other attributes. Other frameworks consider security as a separate category, e.g., in [6]. In our framework, we support non-overlapping monitoring of QoS metrics in which any measured value should be assigned to a single class. Therefore, we believe that *Security* is better classified as a subclass of *Dependability*. Security metrics, as well-known in literature, include *Authentication, Authorization, Confidentiality, Integrity, SecurityLevel*

and *Accountability*. *SecurityLevel* specifies whether security is ensured at message or transport levels which are widely used in Web/WSRF services [30, 2].

The *Configuration* subclass includes several metrics indicating the configuration status. *VirtualOrganization* identifies the Grid virtual organization to which a Grid service belongs. The *Location* metrics are based on QoS *LocationAffinity*, introduced in [18], that reflect the location on which a service resides or is executed. *LevelOfService* is defined in [31] that includes *best effort* and *guaranteed* services. *ServiceVersion* indicates the software version of the service whereas *SupportedStandard* specifies standards that a service supports [8, 28].

The *CustomMetric* subclass indicates QoS metrics which are defined by specific services.

3.2 On Providing the QoS Classification

In monitoring these QoS metrics, our approach is to provide an orthogonal QoS classification. However, in practice, in many cases it is impossible to support an orthogonal QoS classification because several QoS metrics are combined ones. For instance, an unavailable service may be due to an unauthorized use or an unreachable problem. It is, however, important that any metric is assigned to only one subclass of the QoS classification.

Note that existing frameworks normally discuss normalized value for QoS metrics, for example, the availability of a service S_i is given as $Availability = 80\%$. However, it is normally not good enough if a framework provides only a single value for a QoS metric. In many cases clients are interested in having detailed information, for example, when is the last of time of the inaccessible status, MTBF, MTTR, etc. Our framework aims at providing not only a single value for a QoS metric but also other detailed information to the client.

Given a service, it does not mean that a QoS monitoring framework will or could provide all QoS metrics specified in the classification. Depending on individual monitored resource, only a subset of QoS metrics might be provided. For example, a network path (see Section 4.2) is not associated with frame-based QoS metrics.

In our view, we can have different values, provided by the monitoring framework, for QoS attributes of a single monitored resource. It is due to the fact that clients are disparate in the Grid and their views to the monitored resource are different. For example, to a client the availability of a monitored resource is 100%, but to another one it is only 90% because the network path from the second one to the monitored resource is not fully available during the requested time. Therefore, QoS metrics for Grid have to be determined based on *client local view* or *system global view*.

4 Architecture of QoS Monitoring and Analysis Framework

4.1 System Overview

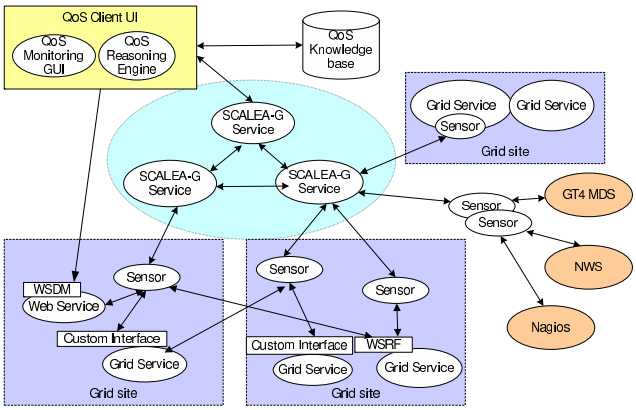

Figure 2. Architecture of QoS monitoring and analysis framework

Figure 2 depicts the architecture of our QoS monitoring and analysis framework. The main components include *monitoring sensors*, *monitoring middleware*, *QoS Client UI*, and *QoS Knowledge base*. In order to monitor Grid services we use distributed *sensors*. Basically, sensors collect monitoring data (usually in the form of events and profiling data) for determining QoS metrics. Specific sensors can conduct QoS analysis and provide QoS metrics for specific resources. The *monitoring middleware* is based on the SCALEA-G framework - a unified performance monitoring and data integration for the Grid [33]. SCALEA-G provides a peer-to-peer Grid infrastructure of monitoring services that allows clients in the Grid to publish and retrieve various types of monitoring data. QoS metrics collected by sensors for individual services are sent to the SCALEA-G middleware which stores monitoring data in distributed monitoring services. Our sensors can also interface to existing monitoring services such as Globus MDS (Monitoring and Discovery System) [3], NWS (Network Weather Service) [35], Ganglia [23], and Nagios [5], using available monitoring data in these services for monitoring and analyzing QoS attributes.

The *QoS Client UI* includes a GUI and a QoS reasoning engine. The *QoS Knowledge base* contains analysis rules for specific metrics and resources, dependencies between monitored resources, and historical QoS data resulted from previous analyses. The QoS monitoring GUI allows the user to conduct the QoS monitoring and analysis of Grid services online. The user can also model dependencies among Grid services and based on that further analysis can be done.

The *QoS reasoning engine* performs QoS analysis, based upon the rules stored within the knowledge base. Automatic rules can be used to react to system changes by alerting the client or even invoking management interfaces (e.g., using WSDM) of services to correct failures.

4.2 Monitored Resources and Measurement Methods

The measurement methods are dependent on types of monitored resources and on QoS metrics.

We classify types of monitored resources into *machine, network path, middleware* and *application*. Machines are places on which middleware and applications are deployed and executed. Network path is a connection between two end points which can be a machine or middleware or applications. Since our supporting Grid applications are based on Web/WSRF services and workflows of Web/WSRF services, the middleware, network path and machine studied are those involved in the execution of Web/WSRF services. Applications and middleware services can implement standard interfaces, e.g. WSDM, used for service monitoring and management, but they also can provide specific interfaces for monitoring purposes. For each type of resources, we apply different monitoring mechanisms to evaluate QoS attributes. For example, Table 1 presents a few resources and corresponding measurement methods. By utilizing WS/WSDM/WSRF interfaces, sensors can remotely monitor Grid applications or middleware providing specific functionality like file transfers and job executions. Instrumentation can also be used to collect performance data. Moreover, we are developing message-level techniques in which SOAP messages are automatically generated and sent to remote WS. Based on SOAP responses of WS, we can determine some QoS metrics.

Monitored Resources	*Measurement Methods*	*Metrics*
machine	using ping	availability
network path	using TCP connection, ping	availability, reliability
middleware	using GRAM, GridFTP, log files	availability, reliability
application	WS/WSRF/WSDM interfaces, SOAP message, instrumentation	availability, reliability, manageability, performance

Table 1. Example of monitored resources and measurement methods

Because of different types of monitored resources, a single, unified measurement method will not be adequate. To evaluate performance-based QoS metrics, we extend our research on performance monitoring and analysis for the Grid to measure and monitor QoS *Performance* metrics, for example, investigating several mechanisms to instrument WS/WSRF/WSDM services and to collect performance metrics at runtime. Moreover, we are working on sensors which gather various monitoring data from log files of Grid middleware services, such as Web/WSRF service containers and job submission systems, and from Web proxies/wrappers. For example, Globus GRAM log information can be extracted for determining *ServiceThroughput* attribute.

To measure dependability-based QoS metrics, we have developed a set of sensors to monitor services and to analyze existing log files. For example, Table 2 shows a few examples of service statuses, provided by sensors used to analyze dependability metrics. *Accuracy* is application-specific, therefore, we develop sensors to test accuracy only for specific services, e.g., GridFTP service. We are currently investigating a method in which the monitoring framework can provide interfaces/sensor templates for the client to evaluate the accuracy or to develop real methods to check the accuracy of a service. Accuracy values can then be stored into the monitoring framework. SOAP messages are also used for monitoring reliability and accuracy metrics. For example, the developer can define SOAP request and response messages. These messages are used to test whether WS are available or not, or a WS operation returns an accurate result. *Capacity* will be monitored through accessing properties of monitored resources using WSRF or WSDM interface or by using customized sensors which conduct capacity tests. *Security*, for example, will be monitored based on the analysis of configuration and log files whereas *Manageability* is monitored through accessing resource properties.

For *Configuration* metrics, we have sensors which will provide static information about the location, version, etc., by processing configuration files.

5 Monitoring and Managing QoS Approach

5.1 Monitoring QoS of Individual Grid Services

Each sensor monitors resources and sends its collected data into the SCALEA-G middleware. Our effort is to develop new sensors to monitor QoS metrics of Grid services, especially for WSRF/WS/WSDM services. Since we have various types of resources, we associate each monitored resource with a unique identifier named *resourceID*. Each type of monitoring data is identified by a unique *dataTypeID*. Thus, a tuple (*dataTypeID, resourceID*) is used to determine all monitoring data of type *dataTypeID* associated with the monitored resource *resourceID*. Table 3 shows examples of resource identifiers. Our framework is extensi-

Monitoring status	*Description*
UP	a value of *Availability* metric that indicates a Grid service is operating normally and able to perform its functional tasks. This value is similar to operational status `Available` in MUWS [7].
DOWN	a value of *Availability* metric that indicates a Grid service is not operating and is not able to perform any function tasks. The service may have been stopped or failed. This value is similar to operational status `Unavailable` in MUWS.
UNKNOWN	a value of QoS attributes that indicates a Grid service is unable to report QoS status at this time. Similar to operational status `Unknown` in MUWS.
UNREACHABLE	a value of *Accessibility* indicates that a Grid service is operating but the client cannot reach the service due to some problems on the path from client to service side.
UNACCESSIBLE	a value of *Accessibility* indicates that a Grid service is operating but the client cannot access it due to some problems. For example, the number of requests exceeds the service threshold.

Table 2. Example of monitoring statuses

Resource	*ResourceID*
IP network path	icmp://pleisen.dps.uibk.ac.at->zeus72.cyf-kr.edu.pl
TCP server	tcp://altix1.uibk.ac.at:22
GRAM	gram://altix1.uibk.ac.at/jobmanager-pbs
GridFTP	gridftp://altix1.uibk.ac.at/
WS, WSRF, WSDM	http://zeus72.cyf-kr.edu.pl:8080/wsrf/services/gom/service/GOMService?wsdl

Table 3. Examples of resource identifiers

ble and it provides sensor templates, thus, any new sensors can be easily developed for monitoring new resources.

5.2 QoS Monitoring and Analysis of Dependent Services

We develop an integrated GUI and a QoS reasoning engine for examining QoS metrics of dependent services, supporting modeling and monitoring dependencies. Dependencies among Grid services are modeled as a graph in which a node represents a Grid service and an edge between two nodes represents a dependency. Nodes are single Grid services which can be monitored, and the status of nodes is provided either directly by sensors or indirectly through the analysis of dependencies among nodes. Each node is associated with a set of QoS attributes. Edges describe dependencies among monitored resources. A dependency can be a *casual* relationship or *mutually exclusive* one. For example, a Grid workflow execution service can have casual relationships to a file transfer service and a job manager. If the file transfer service or the job manager is `DOWN`, the execution service will not be able to fulfil requests from clients. Therefore, the execution service can be monitored indirectly through the monitoring of the file transfer service and the job manager (both monitored directly). On the other hand, a workflow can have mutually exclusive relationships to two similar WS. The two similar WS provide the same features so the workflow can use any one of them. If one WS is `DOWN`, the other can be used. Thus, the workflow fails when both WS are not operational. The above-mentioned examples show that both types of dependencies are crucial. A relationship can be associated with a *functional* property which indicates that the cause will affect only to functional tasks, not operating mode. For example, while the file transfer service is `DOWN`, the execution service is `UP`, but not functional properly.

In our tool, most casual relationships can automatically be detected, based on monitoring data. However, some casual with functional property and most mutually exclusive relationships are service/application specific, thus they are not detected by the tool. Therefore, we support the user to define such relationships as well as subsets of dependent services which will be monitored and analyzed by the tool.

5.3 Storing and Forecasting QoS metrics of Grid services

Based on QoS metrics of individual services and their dependencies, we develop techniques to predict QoS metrics of Grid services. In doing so, the QoS knowledge base is also used to store analysis results. The online monitoring and analysis of QoS, based on real time monitoring data, produces resulting QoS metrics which are stored in the QoS knowledge base for later use. Based on the data in the knowledge base, other middleware services, e.g., the scheduler, can query historical QoS data of any Grid services. Currently, our QoS knowledge base is being developed as a WSRF service. Moreover, based on historical data, various techniques can be employed to conduct forecasts on QoS metrics of monitored resources. Such forecasts are useful inputs for selecting resources and establishing service agreements between clients and service providers.

6 Experiments

Our current implementation, based on GT 4.0 [17] and integrated into SCALEA-G, supports monitoring and analyzing QoS metrics of individual and dependent Grid services, a QoS WSRF service providing QoS metrics, but not forecasting QoS metrics of Grid services (discussed in Section 5.3). Monitoring and managing WSDM services is based on Apache MUSE [1], but this part of work has not been fully achieved. In this section, we present an experiment conducted in the AustrianGrid [10] and K-WfGrid [4]

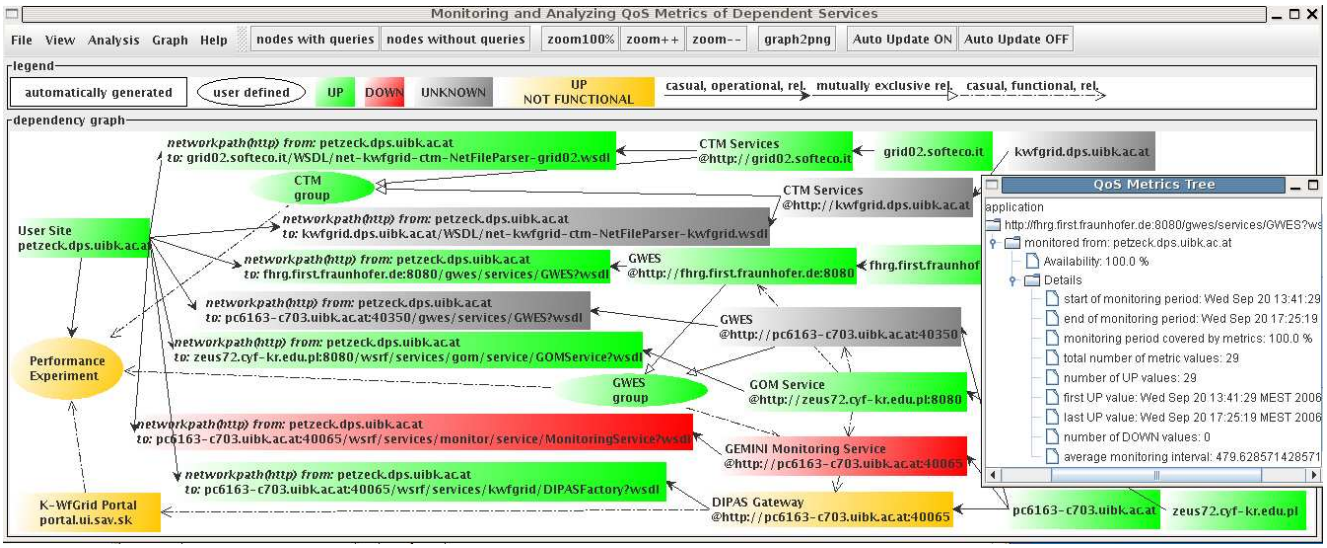

Figure 3. Example of monitoring dependent Grid services.

testbeds.

Figure 3 presents an example in which we monitored and analyzed QoS metrics of all services involved in a performance experiment. This performance experiment is used to conduct the performance analysis for a workflow named `CTM` in the K-WfGrid testbed. In doing so, from `petzeck.dps.uibk.ac.at`, we used the K-WfGrid portal at `portal.ui.sav.sk` to submit the workflow to the GWES (Grid Workflow Execution Service) deployed in `fhrg.first.fraunhofer.de` and `pc6163-c703.uibk.ac.at`, and conducted the performance analysis. In order to execute workflows, GWES is dependent on GOM which is deployed in `zeus72.cyf-kr.edu.pl`. The workflow can be monitored and analyzed only if the DIPASGateway is available. DIPASGateway is dependent on GOM and GEMINI whereas GEMINI is dependent on GWES. Both GEMINI and DIPASGateway are dependent on GOM. While our tool can automatically detect the causal relationships between `petzeck.dps.uibk.ac.at` and relevant services such as GWES, Portal, GOM, GEMINI and DIPASGateway, it does not have any information about the interdependency among these services. Thus, we had to manually add these relationships. The `CTM` workflow can be executed only if `CTM` Web services in `grid02.softeco.it` or in `kwfgrid.dps.uibk.ac.at` are working. Therefore, we defined a `CTM group` which represents `CTM` services. `CTM group` has mutually exclusive relationships to the two deployments of `CTM` services. Since we have two instances GWES, a `GWES group` which has mutually exclusive relationships to GWES instances is defined. The performance experiment, defined as a node named `Performance Experiment`, is considered to be dependent on `petzeck.dps.uibk.ac.at` machine, `CTM group`, `GWES group`, and the K-WfGrid Portal. Before running the performance experiment, by using this tool, we can check if all services involved in the intended experiment are available. For example, in Figure 3, GEMINI was `DOWN`. Consequently, due to casual, functional, relationships, DIPASGateway and K-WfGrid Portal were `UP`, but not functional properly for the performance experiment. Therefore, the performance experiment could not be conducted. Different colors indicate different statuses of monitored resources. Also, during the experiment, status of services is updated and we can retrieve QoS metrics of every services. For example, the dialog `QoS Metrics Tree` displayed the Availability metric of the GWES deployed in `fhrg.first.fraunhofer.de`. The above-mentioned experiment shows how our tool can simplify the monitoring and analysis of QoS metrics of Grid services with complex dependencies.

7 Conclusion and Future Work

In this paper, we have presented a novel framework for monitoring and analyzing QoS metrics of Grid services. We have also demonstrated our prototype for monitoring and analyzing QoS of various Grid services. The main contributions are the novel classification of QoS metrics and techniques to monitor and analyze dependent Grid services.

We are currently working on the full implementation of the framework. Firstly, we are enhancing the framework by extending sensors and measurement techniques for monitoring QoS metrics. Secondly, we are working on supporting forecasting QoS metrics and on the implementation of QoS

knowledge base for storing QoS metrics.

References

[1] Apache webservices - muse, http://ws.apache.org/muse.

[2] GT 4.0 Security: Message & Transport Level Security, http://www-unix.globus.org/toolkit/docs/4.0/security/message/.

[3] GT Information Services: Monitoring & Discovery System (MDS), http://www.globus.org/toolkit/mds/.

[4] K-WF Grid Project. http://www.kwfgrid.net.

[5] Nagios, http://www.nagios.org/.

[6] QoS for Web Services: Requirements and Possible Approaches, http://www.w3c.or.kr/kr-office/tr/2003/ws-qos/.

[7] Web Services Distributed Management: Management Using Web Services (MUWS 1.0) Part 2, http://docs.oasis-open.org/wsdm/2004/12/wsdm-muws-part2-1.0.pdf.

[8] Understanding quality of service for Web services, http://www-128.ibm.com/developerworks/library/ws-quality.html, Jan 2002.

[9] R. J. Al-Ali, K. Amin, G. von Laszewski, O. F. Rana, D. W. Walker, M. Hategan, and N. J. Zaluzec. Analysis and provision of qos for distributed grid applications. *J. Grid Comput.*, 2(2):163–182, 2004.

[10] AustrianGrid. http://www.austriangrid.at/.

[11] A. Avizienis, J. Laprie, and B. Randell. Fundamental concepts of dependability. Technical Report N01145, LAAS-CNRS, 2001.

[12] S. N. Bhatti, S.-A. Sorensen, P. Clark, and J. Crowcroft. Network QoS for Grid Systems. *International Journal of High Performance Computing Applications*, 17(3):219–236, 2003.

[13] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An approach for qos-aware service composition based on genetic algorithms. In *GECCO*, pages 1069–1075. ACM, 2005.

[14] J. Cardoso, A. P. Sheth, J. A. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *J. Web Sem.*, 1(3):281–308, 2004.

[15] G. Dobson, R. Lock, and I. Sommerville. Qosont: a qos ontology for service-centric systems. In *EUROMICRO-SEAA*, pages 80–87. IEEE Computer Society, 2005.

[16] D. Firesmith. Specifying reusable security requirements. *Journal of Object Technology*, 3(1):61–75, 2004.

[17] Globus Project. http://www.globus.org.

[18] I.Brandic, S. Pllana, and S. Benkner. High-level Composition of QoS-aware Grid Workflows: An Approach that Considers Location Affinity. In *HPDC 2006, Workshop on Workflows in Support of Large-Scale Science (WORKS06)*. IEEE Computer Society, June 2006.

[19] C. E. Irvine and T. E. Levin. Toward a taxonomy and costing method for security services. In *ACSAC*, pages 183–188. IEEE Computer Society, 1999.

[20] M. C. Jaeger, G. Rojec-Goldmann, and G. Mühl. Qos aggregation for service composition using workflow patterns. In *Proceedings of the 8th International Enterprise Distributed Object Computing Conference (EDOC 2004)*, pages 149–159, Monterey, California, USA, September 2004. IEEE CS Press.

[21] J.-K. Kim, T. Kidd, H. J. Siegel, C. E. Irvine, T. E. Levin, D. A. Hensgen, D. S. John, V. K. Prasanna, R. F. Freund, and N. W. Porter. Collective value of qos: A performance measure framework for distributed heterogeneous networks. In *IPDPS*, page 84. IEEE Computer Society, 2001.

[22] Y. Liu, A. H. H. Ngu, and L. Zeng. Qos computation and policing in dynamic web service selection. In *WWW (Alternate Track Papers & Posters)*, pages 66–73. ACM, 2004.

[23] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, May 2004.

[24] D. A. Menascé, H. Ruan, and H. Gomaa. A framework for qos-aware software components. In *WOSP*, pages 186–196. ACM, 2004.

[25] S. B. Musunoori, F. Eliassen, and R. Staehli. Qos-aware component architecture support for grid. In *Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2004), Modena, Italy*, pages 277–282. IEEE Computer Society Press, 2004.

[26] I. V. Papaioannou, D. T. Tsesmetzis, I. G. Roussaki, and M. E. Anagnostou. A qos ontology language for web-services. *AINA*, 1:101–106, 2006.

[27] C. Patel, K. Supekar, and Y. Lee. A qos oriented framework for adaptive management of web service based workflows. In *DEXA*, volume 2736 of *Lecture Notes in Computer Science*, pages 826–835. Springer, 2003.

[28] S. Ran. A model for web services discovery with qos. *SIGecom Exch.*, 4(1):1–10, 2003.

[29] F. Rosenberg, C. Platzer, and S. Dustdar. Bootstrapping performance and dependability attributes of web services. In *IEEE International Conference on Web Services (ICWS'06)*, 2006.

[30] J. Rosenberg and D. Remy. *Securing Web Services with WS-Security : Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*. Sams, 2004.

[31] B. Sabata, S. Chatterjee, M. Davis, J. J. Sydir, and T. F. Lawrence. Taxomomy of qos specifications. In *WORDS '97: Proceedings of the 3rd Workshop on Object-Oriented Real-Time Dependable Systems - (WORDS '97)*, page 100, Washington, DC, USA, 1997. IEEE Computer Society.

[32] R. Sumra and A. D. Quality of Service for Web Services: Demystification, Limitations, and Best Practices, http://www.developer.com/services/article.php/202791.

[33] H. L. Truong and T. Fahringer. Self-managing sensor-based middleware for performance monitoring and data integration in grids. In *IPDPS*. IEEE Computer Society, 2005.

[34] H. L. Truong, T. Fahringer, F. Nerieri, and S. Dustdar. Performance metrics and ontology for describing performance data of grid workflows. In *CCGRID*, pages 301–308. IEEE Computer Society, 2005.

[35] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computing Systems*, 15:757–768, 1999.

[36] OASIS Web Services Distributed Management (WSDM) TC, http://www.oasis-open.org/committees/wsdm/ipr.php.

[37] J. Yu, R. Buyya, and C.-K. Tham. Cost-based scheduling of scientific workflow application on utility grids. In *e-Science*, pages 140–147. IEEE Computer Society, 2005.