Appendix to the Deliverable D2.5.1 Specification and Design of the Basic System Components

Performance Monitoring and Analysis for Grid Infrastructures and Applications

WP3

Document Filename: KWF-WP3-D2.5.1-UNIINS-v1.2-GridPMAService
Workpackage: WP3
Partner(s): UNIINS, CYF, FIRST
Lead Partner: UNIINS
Document classification: CONFIDENTIAL

Abstract: This appendix to the main deliverable D2.5.1 specification and design of the basic system components describes the specification and design of Grid performance monitoring and analysis services in K-WfGrid project. We present an overview of the architecture of the performance and monitoring services and discuss useful performance and dependability metrics for workflows in K-WfGrid. We describe basic system components including the monitoring and instrumentation service, the performance analysis service, data representations and service interfaces. Finally, we outline interdependencies between WP3 and other WPs in K-WfGrid.
Delivery Slip

<table>
<thead>
<tr>
<th>Name</th>
<th>Partner</th>
<th>Date</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>From Hong-Linh Truong, Thomas Fahringer</td>
<td>UNIINS</td>
<td>29/06/2005</td>
<td></td>
</tr>
<tr>
<td>Verified By Piotr Nowakowski</td>
<td>CYFRONET</td>
<td>30/06/2005</td>
<td></td>
</tr>
<tr>
<td>Approved By Steffen Unger</td>
<td>FIRST</td>
<td>30/06/2005</td>
<td></td>
</tr>
</tbody>
</table>
## Document Log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Summary of changes</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>12/9/2004</td>
<td>Skeleton of version 0.1.</td>
<td>Hong-Linh Truong</td>
</tr>
<tr>
<td></td>
<td>26/9/2004</td>
<td>Update architecture of Performance Analysis Service</td>
<td>Hong-Linh Truong, Thomas Fahringer</td>
</tr>
<tr>
<td></td>
<td>1/10/2004</td>
<td>First version of Monitoring Service architecture chapter</td>
<td>Bartosz Balis, Hong-Linh Truong</td>
</tr>
<tr>
<td></td>
<td>20/10</td>
<td>Outline of information that WP3 can provide WP2</td>
<td>Hong-Linh Truong, Bartosz Balis</td>
</tr>
<tr>
<td></td>
<td>24/10/2004</td>
<td>Dependencies between WP4 and WP3, Update dependencies between WP3 and WP3</td>
<td>Hong-Linh Truong</td>
</tr>
<tr>
<td></td>
<td>26/10/2004</td>
<td>First version of data representation chapter, outline the event-infrastructure</td>
<td>Hong-Linh Truong</td>
</tr>
<tr>
<td></td>
<td>28/10/2004</td>
<td>Refine event infrastructure, add glossary and abbreviation chapter</td>
<td>Bartosz Balis, Hong-Linh Truong</td>
</tr>
<tr>
<td></td>
<td>1/11/2004</td>
<td>Update and refinement of Usage Scenarios</td>
<td>Bartosz Balis, Hong-Linh Truong</td>
</tr>
<tr>
<td>0.2</td>
<td>2/11/2004</td>
<td>Release version 0.2</td>
<td>Hong-Linh Truong (editor)</td>
</tr>
<tr>
<td></td>
<td>24/11/2004</td>
<td>Performance metrics of workflows</td>
<td>Hong-Linh Truong</td>
</tr>
<tr>
<td></td>
<td>6/12/2004</td>
<td>Update architecture, performance analysis service</td>
<td>Hong-Linh Truong, Thomas Fahringer, Francesco Neri, Radu Prodan</td>
</tr>
<tr>
<td></td>
<td>7/12/2004</td>
<td>Update architecture of Monitoring Service, description of sensors</td>
<td>Bartosz Balis</td>
</tr>
<tr>
<td>0.3</td>
<td>14/12/2004</td>
<td>Release version 0.3</td>
<td>Hong-Linh Truong, Thomas Fahringer (editor)</td>
</tr>
<tr>
<td></td>
<td>7/01/2005</td>
<td>Added description of infrastructure monitoring system</td>
<td>Bartosz Balis</td>
</tr>
<tr>
<td></td>
<td>10/01/2005</td>
<td>Updated and extended descriptions in monitoring service chapter: application monitoring, infrastructure monitoring, performance repository. Figures added.</td>
<td>Bartosz Balis</td>
</tr>
<tr>
<td></td>
<td>14/01/2005</td>
<td>Update performance analysis chapter</td>
<td>Hong-Linh Truong, Radu Prodan</td>
</tr>
<tr>
<td></td>
<td>17/01/2005</td>
<td>Update architecture chapter</td>
<td>Hong-Linh Truong, Thomas Fahringer</td>
</tr>
<tr>
<td></td>
<td>19/01/2005</td>
<td>Update introduction, data representation, service interface and workpackage dependencies</td>
<td>Hong-Linh Truong</td>
</tr>
<tr>
<td></td>
<td>12/03/2005</td>
<td>Major update of monitoring chapter, text and figures.</td>
<td>Bartosz Balis</td>
</tr>
<tr>
<td></td>
<td>14/03/2005</td>
<td>Update event infrastructure and workflow level instrumentation</td>
<td>Andreas Hoheisel, Tilman Linden</td>
</tr>
<tr>
<td></td>
<td>14/03/2005</td>
<td>Add PDQS schema and update data query and subscription section</td>
<td>Hong-Linh Truong</td>
</tr>
<tr>
<td>0.4</td>
<td>16/03/2005</td>
<td>Revise version 0.3 and release version 0.4</td>
<td>Hong-Linh Truong, Thomas Fahringer</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Summary of changes</td>
<td>Author</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>0.5</td>
<td>17/06/2005</td>
<td>Add SIRWF and WIRL sections, update data representation</td>
<td>Hong-Linh Truong</td>
</tr>
<tr>
<td></td>
<td>21/06/2005</td>
<td>Update data schemas in appendix</td>
<td>Hong-Linh Truong</td>
</tr>
<tr>
<td></td>
<td>29/06/2005</td>
<td>Release Version 0.5</td>
<td>Thomas Fahringer, Hong-Linh Truong, Bartosz Balis, Andreas Hoheisel, Tilman Linden</td>
</tr>
<tr>
<td></td>
<td>03/08/2005</td>
<td>Fix some minor mistakes</td>
<td>Hong-Linh Truong</td>
</tr>
</tbody>
</table>
## Contents

1 INTRODUCTION

1.1 UNIFIED SYSTEM FOR PERFORMANCE MONITORING AND ANALYSIS ........................................ 7
1.2 DYNAMICALLY ENABLED INSTRUMENTATION OF MULTILINGUAL APPLICATIONS ....................... 8
1.3 SEMANTIC, WELL-DEFINED DATA REPRESENTATIONS AND SERVICE INTERFACES ..................... 9
1.4 MONITORING AND PERFORMANCE ANALYSIS OF GRID WORKFLOWS .................................. 9
1.5 OUTLINE OF THIS REPORT ....................................................................................................... 10

2 SUPPORTING GRID WORKFLOW-BASED APPLICATIONS AND INFRASTRUCTURE ..................... 11

2.1 WORKFLOW MODEL ............................................................................................................... 11
2.2 GRID INFRASTRUCTURE ........................................................................................................ 11
2.3 EXECUTION MODEL OF WORKFLOWS .................................................................................... 12
2.4 MONITORED RESOURCES AND MONITORING SENSORS .................................................... 12

3 ARCHITECTURE .......................................................................................................................... 13

3.1 PERFORMANCE MONITORING AND ANALYSIS IN K-WFGrid .............................................. 13
3.2 MONITORING AND PERFORMANCE ANALYSIS SCENARIO .................................................. 14
3.3 WP3 SERVICE ARCHITECTURE ............................................................................................... 15

4 EVENTS, PERFORMANCE AND DEPENDABILITY METRICS OF WORKFLOWS ....................... 16

5 GRID MONITORING AND INSTRUMENTATION SERVICE .......................................................... 19

5.1 ARCHITECTURE .................................................................................................................... 19
5.2 GENERIC SENSOR LIBRARY ................................................................................................. 22
5.3 SENSORS ............................................................................................................................... 22
5.4 INTERACTION WITH CLIENTS ............................................................................................... 23
5.5 APPLICATION MONITORING ............................................................................................... 23
5.6 GRID INFRASTRUCTURE MONITORING ............................................................................... 25
5.7 EVENT INFRASTRUCTURE ..................................................................................................... 26
5.8 PERFORMANCE DATA REPOSITORY ...................................................................................... 27

6 GRID PERFORMANCE ANALYSIS SERVICE ......................................................................... 29

6.1 DISTRIBUTED PERFORMANCE ANALYSIS SERVICE (DIPAS) .............................................. 29
6.2 PERFORMANCE INTERPRETATION AND BOTTLENECK SEARCH ....................................... 30
6.3 PERFORMANCE EXPERIMENT REPOSITORY ....................................................................... 31
6.4 PERFORMANCE VISUALIZATION ......................................................................................... 31
# PERFORMANCE SERVICE INTERFACES AND DATA REPRESENTATIONS

## 7.1 STANDARDIZED INTERMEDIATE REPRESENTATION FOR INVOKED APPLICATIONS

## 7.2 WORKFLOW INSTRUMENTATION REQUEST LANGUAGE

## 7.3 DATA REPRESENTATION

## 7.4 PERFORMANCE DATA QUERY AND SUBSCRIPTION REQUEST

## 7.5 SERVICE OPERATIONS

# WORKPACKAGES DEPENDENCIES

## 8.1 DEPENDENCIES BETWEEN WP3 AND WP2

## 8.2 DEPENDENCIES BETWEEN WP3 AND WP4

## 8.3 DEPENDENCIES BETWEEN WP3 AND WP5

# ABBREVIATIONS

## SIRWF SCHEMA

## WORKFLOW INSTRUMENTATION REQUEST LANGUAGE

### C.1 WIRL SCHEMA

### C.2 WIRL REQUEST NAMES

### C.3 INSTRUMENTATION LOCATIONS

## DATA REPRESENTATION

### D.1 COMMON SCHEMAS

### D.2 XML SCHEMAS OF MONITORING DATA

## PERFORMANCE DATA QUERY AND SUBSCRIPTION LANGUAGE

### E.1 PDQS SCHEMA

### E.2 PDQS Data Aggregation
1 INTRODUCTION

The objective of this workpackage is to build Grid services for monitoring and performance analysis of Grid infrastructures and workflow-based Grid applications. The services will be unique by integrating the instrumentation, monitoring and performance analysis of the Grid infrastructures and applications into a unified system following the ideas of the Grid Monitoring Architecture (GMA) [15], OGSA [22], and the peer-to-peer model [34].

The work will be based on a common XML-based representation for instrumentation and monitoring requests as well as on XML-represented performance and event data which can be used by other services (e.g., scheduling services, knowledge-based systems), user interfaces and Grid applications to invoke and control the monitoring and performance analysis.

This report describes the specification of basic system components of the Grid monitoring and performance analysis services. It analyzes existing instrumentation and performance analysis technologies, and describes the basic design of software components of this workpackage.

1.1 UNIFIED SYSTEM FOR PERFORMANCE MONITORING AND ANALYSIS

Most existing Grid monitoring tools are separated into two distinct domains based on what they are monitoring: Grid infrastructure monitoring and Grid application monitoring (see [23, 54] for a greater detail). However, in a Grid system, which by nature is a very complex, various resources and services coexist and they introduce sophisticated interactions. Thus, the lack of the combination of two domains in a single system has hindered the user from correlating measurement metrics from various sources at different levels during monitoring and performance analysis. Grid monitoring tools that are able to combine application and system monitoring and performance analysis are crucial as these tools will provide the user a unified view and support the correlation between performance metrics from various sources. In addition, most existing Grid monitoring tools focus on the monitoring and analysis for Grid infrastructure; yet little effort has been done on performance analysis for Grid applications, especially for Grid workflows based on a service-oriented architecture (SOA).

To date, most existing Grid performance analysis tools are not based on Grid service technologies, as presented in [23], because most of them are originally designed and developed for conventional parallel and distributed systems. As a result, these performance analysis tools do not well address challenges in Grid environment such as scalability, diversity, dynamics and security, and are not easily integrated into and deployed in Grid service-based infrastructures. Performance measurement, instrumentation and analysis for Grid applications require different approaches from those for conventional parallel systems. Moreover, Grid monitoring and performance analysis services not only supports the end-user to monitor and analyze Grid infrastructures and applications but it also provides useful information for several other services such as workflow optimization and scheduling services. Therefore, Grid monitoring and performance analysis services must provide well-defined interfaces and describe performance and monitoring data with a well-defined representation so that other services can easily access and monitor performance data.

Instead of monitoring applications or infrastructure separately, we propose a unified approach to the performance monitoring and analysis for the Grid [46]. That is, the performance of both applications and infrastructure should be monitored and analyzed in an integrated system [46]. Also, Grid monitoring middleware must be robust and adaptive to the dynamics of the Grid [46]. This requires us to investigate new novel architecture for Grid monitoring tools, for example based on peer-to-peer model [34], self-organizing architecture [30], and adaptive sensor networks [13, 49]. R-GMA [9], developed in DataGrid [44], supports both monitoring and information systems, storing generic monitoring data, but not addressing adaptive monitoring issues. R-GMA uses relational database and this results in slow access to monitoring data. The Institute on Grid Information and Monitoring Services
of the CoreGrid project [1] aims at providing a generic and scalable monitoring and information system. Thus, common ideas and techniques, e.g., peer-to-peer model and self-organizing architecture, can be employed in both K-WfGrid and CoreGrid even though at the time of writing this report we do not know which technologies will be used in CoreGrid. In K-WfGrid, we will develop a unified system that combines application and system monitoring and performance analysis for workflows of Grid services. Moreover, in order to support the interoperability among different Grid services, the performance monitoring and analysis services will be built based on a service oriented architecture (SOA).

1.2 DYNAMICALLY ENABLED INSTRUMENTATION OF MULTILINGUAL APPLICATIONS

Today, workflow-based applications are often multilingual written, for instance, in C/C++/Fortran and Java. In fact, most multilingual applications fall into two popular categories:

- Java-based applications execute C/C++/Fortran code via the exec mechanism (by using java.lang.Process). In this category, C/C++/Fortran codes are in the form of executable programs.
- Java-based applications wrap C/C++/Fortran code via JNI (Java Native Interface). In this category, C/C++/Fortran codes are provided in the form of shared libraries.

However, little effort has been spent on investigating the instrumentation for multilingual applications. The CrossGrid project [2] supports instrumentation and online applications monitoring, mostly for MPI applications, but not for workflows and multilingual applications. We will support instrumentation of multilingual applications, especially in the case where C/C++/Fortran code is called by Java because this type of multilingual applications is widely employed in practice.

Currently, the three most common instrumentation approaches are source code instrumentation, binary wrapping (or library-based wrapping) and dynamic instrumentation (runtime patching). Source code instrumentation usually relies on tools which parse target source code and insert code snippets into appropriate places. The drawback of this method is that we have to change the source code of the application and to recompile it each time the instrumentation is conducted. The advantage is that we can instrument the code at practically arbitrary places (e.g., loops and user-defined code regions).

Binary wrapping [18] implies the modification of symbol tables of object files. Using this method, we rename some function names and provide corresponding wrapper functions with the original name. A wrapper function calls the original one while performing some additional activities before and after the call. Binary wrapping has the advantage that no modification of application's source code is necessary. In this method we can prepare pre-instrumented versions of common libraries, such as MPI, and only relink the application with the instrumented library, when necessary. The disadvantage of this method is that we can only instrument the code at the level of function calls. Moreover, this method is normally applied to third party libraries.

A few frameworks support dynamic instrumentation, such as DynInst [3]. This method allows the dynamic insertion of instrumentation code directly into program’s memory, at runtime. Though this method offers the highest flexibility in terms of dynamic instrumentation, it is not highly portable. Moreover, dynamic instrumentation mostly supports to instrument code at the granularity of function calls.

Another approach is to combine source code instrumentation and/or binary wrapping with the dynamic control of the measurement process at runtime. That is source code instrumentation and binary wrapping are used to insert the instrumentation statically and the measurement is controlled dynamically, e.g., to enable/disable the measurement. This can be simply achieved by adding a condition clause wrapping the instrumentation code. The condition is evaluated based on some values which can be easily changed, even by an external process, through a trace mechanism.

For our goals, we plan to use dynamically enabled instrumentation using binary wrapping and, if necessary, the source code instrumentation. This approach is both lightweight and flexible. The overhead of inactive instrumentation (only condition is checked but evaluated to false) is close to zero. Therefore, we can insert the instrumentation
code permanently into an application without deteriorating performance. At runtime we can dynamically enable or disable instrumentation where necessary. For the purpose of performance analysis of workflows we do not need any finer grained instrumentation than function level instrumentation. Binary wrapping is sufficient. This approach is lightweight and similar flexible as dynamic instrumentation.

In case of multilingual applications, we can use the above-mentioned approach to instrument the legacy code wrapped by Java calls, and, additionally, we can exploit the dynamic instrumentation mechanism using the Java Virtual Machine Tool Interface (JVMTI) [6] to instrument the Java calls.

Even though dynamically enabled instrumentation mostly supports measuring performance metrics of program units, and function calls but not of arbitrary code regions, for Grid workflow applications, we believe that measuring the performance at a level of program unit and function call should be enough. One reason is that the Grid workflow application developer may not write the code for specific activities but the developer puts existing components together to create a workflow. We believe that when analyzing the performance of workflow applications, most users are interested in observing the performance at the level of workflow, program unit and function call, rather than at the loop level.

1.3 SEMANTIC, WELL-DEFINED DATA REPRESENTATIONS AND SERVICE INTERFACES

To simplify the integration between different components in a very complex Grid system, both monitoring and performance data and service interfaces have to be well-defined. GLUE initiative [45] works on a uniform schema for different monitoring data, but focusing only computing and storage resources. In K-WfGrid, we will not concentrate on describing computing and storage resources. Instead, K-Wfgrid WP3 will focus on schemas for application monitoring data and languages used to request and control instrumentation and data query and subscription.

We note that knowledge-based description for monitoring data is not on the focus of most performance tools. But consider the complexity of Grid workflows, metrics have to be well-described for extracting knowledge from monitoring data. Given the complexity of Grid workflows, we must consider performance metrics associated with many levels of abstraction. Moreover, we focus on the development and analysis of performance properties of workflows. Our performance analysis will be based on a novel ontology describing performance data of Grid workflows. With respect to the semantic description for monitoring data, the OntoGrid project [7] also uses knowledge gained from monitoring data to debug workflows. Performance analysis and debugging all require the collection of different types of monitoring data but they have very different objectives. The monitoring service developed by K-WfGrid, of course, can be used by OntoGrid.

1.4 MONITORING AND PERFORMANCE ANALYSIS OF GRID WORKFLOWS

Although there exist a variety of well known performance tools such as Parodyn [33], Pablo [40], Paraver [50], EXPERT [20], these tools are neither designed and developed for the Grid nor for monitoring and performance analysis of workflows. Most efforts to support the scientist in developing Grid workflow-based applications focus on workflow languages, workflow construction and execution systems. Rather little work has been devoted to monitoring and performance analysis of the Grid workflows. P-GRADE [29] is one of the few tools that supports tracing of workflow applications. However, P-GRADE is limited to MPI and PVM applications executed on a single Grid site.

Our performance monitoring and analysis service will differ from these tools in many aspects. Firstly, our service will be an OGSA-based service while existing monitoring and performance analysis tools are not. Secondly, we support dynamically enabled instrumentation of Grid workflow-based applications. Our Grid workflow monitoring and performance analysis service will combine online monitoring execution of activities with online profiling analysis of invoked applications. The support of dynamically enabled instrumentation will be applied to a wide
range of applications. Our monitoring and performance analysis techniques target Grid workflows that are based on service-oriented architecture.

In business domain, monitoring of workflows has been discussed for many years. Many techniques have been introduced to study quality of service and performance models of Web services and workflows [31, 17], and to support monitoring and analysis of the execution of the workflow on distributed systems [39, 12]. In Grid domain, rather little work has been devoted to monitoring and performance analysis of the Grid workflows, let alone the monitoring and analysis of workflows of Grid services. We will support dynamically enabled instrumentation of activity instances as well as monitoring and performance analysis of workflows. We will cover not only workflow execution status monitoring but also performance measurements obtained by instrumenting the invoked application and by accessing resource monitoring data. The monitoring and analysis in most existing tools limited to the activity (task) level. In contrast, our performance monitoring and analysis will not be limited to the level of workflow activities but considers also code regions of invoked applications.

Most performance analysis services obtain performance monitoring data from distributed sources and conduct the analysis at a centralized location. Our approach aims at developing a distributed analysis architecture based on which a distributed performance analysis service is built. We will conduct a comprehensive study of performance metrics which can be used to evaluate the performance of a workflow executed on the Grid. Performance metrics are associated with various levels of abstraction. Moreover, we focus on the development and analysis of performance properties of workflows. Our performance analysis will be based on a novel ontology describing performance data of Grid workflows.

1.5 OUTLINE OF THIS REPORT

The rest of this report is organized as follows: Section 2 describes the workflow-based applications by WP3 support and presents preliminaries. Section 3 describes the overall architecture of WP3 and discusses the execution model of the supported workflows. Events, performance metrics, and performance properties of workflows are presented in Section 4. Section 5 discusses the Grid Performance Monitoring and Instrumentation Service. The Grid Performance Analysis Service is presented in Section 6. We outline the performance service interfaces and data representation in Section 7. The dependencies among WP3 and other WPs are described in Section 8.
2 SUPPORTING GRID WORKFLOW-BASED APPLICATIONS AND INFRASTRUCTURE

2.1 WORKFLOW MODEL

WP3 concentrates on analyzing workflows composed by WP2. The Workflow Management Coalition [51] defines a workflow as “The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules”. In the context of Grid computing the term workflow usually concerns the automation of distributed IT processes. In K-WfGrid the workflow model is based on Petri nets. A Petri net consists of two basic elements: transitions (rectangle) and places (circle). Transitions and places are connected with each other by directed edges (arcs). Transitions are related to activities, whereas places store the state of the workflow by means of tokens.

Workflow regions are certain sub-regions of the workflow that can be used as common control structures, such as sequence, choice, split/join, and iteration. In K-Wf Grid every activity (synonyms: task, job) has a specific level of abstraction ranging from abstract to concrete. An abstract activity contains a very brief description of the task and cannot be executed automatically in the real world without further refinement, whereas a concrete activity is mapped to a specific instance of a Web Service operation. Only concrete activities can be directly executed automatically in the real world. The monitoring service can instrument and monitor concrete activities. The analysis service relates performance data of concrete activities to abstract activities.

Within a Web service operation, external applications (e.g. flood forecasting simulation) written in C/C++/Fortran may be invoked. An execution of an activity is called an activity instance. The Web service operations invoked in activity instances are called invoked applications of activities. An invoked application can be considered as a set of code regions. A code region is a sequence of program statements. Code regions can be entire program units such as functions/subroutines and function/subroutine calls.

Many types of workflow regions are supported, including loops. The workflow representation will be developed by WP2. The performance monitoring and analysis of workflows will be conducted at many levels of detail including:

- workflow and workflow region.
- activity.
- code region.

2.2 GRID INFRASTRUCTURE

A Grid infrastructure consists of a set of Grid sites. A Grid site is comprised of a set of Grid services within a single organization that is utilized as a single, unified computing service. The set of grid services includes both software services and hardware resources. A Grid site consists of a number of computational nodes (or computers) that share the same security infrastructure and local network, and are utilized as a single, unified computing resource; computational nodes in a Grid site are controlled by a single resource management service.

A computational node can be any computing platform, from a single-processor workstation to an SMP (Symmetric Multi-Processor) system.

Each computational node may have single or multiple processor(s). On each computational node, multiple application processes execute, each process may have multiple threads of execution.

A computational node communicates with another node through the network. A network path is the network-level abstraction of a virtual link from/between a computational node to/and another node. There are many network-level abstractions such as IP, TCP and HTTP.
2.3 EXECUTION MODEL OF WORKFLOWS

Figure 2.1 presents the simplified execution sequence of a workflow (WF) in K-WfGrid. The user submits a WF to the workflow management system (WfMS). The WfMS instantiates activities. When executing an activity instance, the WfMS locates the remote Web service and invokes operations of the Web service.

The WfMS in K-WfGrid project comprises many components such as GWES (Grid Workflow Execution Service), WCT (Workflow Composition Tool) and Scheduler (see the overall architecture of K-WfGrid in Figure 3.1).

2.4 MONITORED RESOURCES AND MONITORING SENSORS

Any object that is to be monitored is called monitored resource. In our architecture, monitored resources are workflow activities, code regions, Grid sites, computational nodes, application processes, threads, network paths, etc. Each monitored resource is associated with a unique identifier.

Sensors are used to monitor, measure and gather monitoring and performance data of resources. There are several types of monitoring and performance data, each type is determined by a unique identifier and described by an XML schema.
3 ARCHITECTURE

3.1 PERFORMANCE MONITORING AND ANALYSIS IN K-WFGrid

Figure 3.1 presents the overall layered architecture of the K-WfGrid project. The performance monitoring and analysis services gather monitoring information about Grid resources, instrument and monitor the execution of workflows, and analyze the performance of workflows. As shown in Figure 3.1, performance and monitoring analysis services are low level services that provide monitoring data and performance results for various other services such as Grid Workflow Execution Service (GWES), Knowledge Assimilation Agent (KAA), etc.

Apart from the user who uses performance monitoring and analysis services to monitor and analyze workflows, the other main clients of the performance monitoring and analysis services are:

- The Scheduler: the scheduling part of GWES needs monitoring data in order to select resources at runtime.
- The Knowledge Assimilation Agent: performance results are knowledge which will be stored in the knowledge base for further tasks.

Moreover, services in K-WfGrid rely on an event infrastructure, provided by the performance monitoring and analysis services, for publishing and receiving events.
To cope with the dynamic nature of the Grid, the monitoring and analysis services have to operate in distributed and self-organizing manner. Therefore, the monitoring and analysis services will utilize a peer-to-peer model as part of their architecture. As the key issues of the Grid are integration and interoperability, the Grid services for monitoring and performance analysis of Grid infrastructures and workflows must expose a well-defined interface for other services to access them. Moreover, the K-WiGrid infrastructure is based on a service-oriented architecture. As a result, the monitoring and analysis services must be built based on a Grid service-oriented model, e.g. OGSA. Performance data has to be shared among diverse services and multiple types of data have to be collected and delivered by the monitoring and analysis service. Therefore, it requires a common XML-based representation for instrumentation and monitoring requests, and XML representations for performance and event data that can be used by other services, user interfaces and Grid applications to invoke and control the monitoring and performance analysis. As performance results are stored in a knowledge base, a novel ontology describing performance data of Grid workflows is required.

3.2 MONITORING AND PERFORMANCE ANALYSIS SCENARIO

Figure 3.2 depicts the performance monitoring and analysis scenario. When a workflow-based application, which has a set of activities, is submitted to the Grid Workflow Execution Service (GWES), GWES locates available resources and executes activities of the application on the selected resources. During the runtime of the workflow, GWES sends events about the execution status of workflow activities, e.g. submitted, active, etc., to the Monitoring Service (MS). The Grid Performance Analysis Service (PAS) receives these events, delivered from an event infrastructure, by subscribing for events with MS.

PAS processes and analyzes received events. By analyzing events, PAS determines information about resources on which invoked applications of activities are executed. PAS then can make decision to instrument invoked applications of selected activities by sending instrumentation requests to the Instrumentation Service (IS), asking IS to instrument code regions in invoked applications with selected metrics. IS conducts the instrumentation and MS collects measurement data. PAS then sends data query and subscription (DQS) requests to MS to get monitoring data and performance measurements of applications and infrastructures and then conducts performance analysis and bottleneck search. PAS is not a centralized service but rather it is formed by a set of distributed performance analysis and interpretation services.

![Figure 3.2: Performance monitoring and analysis scenarios.](image-url)

The scenario mentioned above illustrates a dynamic approach for monitoring, instrumentation, and performance
analysis of Grid workflows. The monitoring, instrumentation and performance analysis are based on dynamic data obtained at runtime and are frequently changed and customized during the execution of workflows.

### 3.3 WP3 SERVICE ARCHITECTURE

![High-level architecture of Grid performance monitoring and analysis service.](image)

Figure 3.3 presents the architecture of Grid performance monitoring and analysis services. The architecture includes two main services: Grid Performance Monitoring and Instrumentation Service (MIS) and Grid Performance Analysis Service (PAS). All of them will be OGSA-based services executed on multiple Grid sites. They support the instrumentation, monitoring and performance analysis of Grid workflow-based applications and infrastructures. To implement OGSA-based services we rely on existing implementations of the Web Service Resource Framework (WSRF)[19].

The Grid Performance Monitoring and Instrumentation Service (MIS) is responsible for conducting the instrumentation, collecting performance data from applications and resources and providing that data to the Grid Performance Analysis Service or other external services which require performance and monitoring data. MIS includes Instrumentation Service (supporting dynamically enabled instrumentation of Grid workflow applications) and Monitoring Service (collecting and providing performance measurements).

The Grid Performance Analysis Service (PAS) controls the instrumentation of Grid workflow applications and analyzes performance of applications and infrastructures based on performance and monitoring data provided by MIS. Moreover, PAS supports the performance interpretation and bottleneck search for workflows.

MIS and PAS publish information about themselves and about types of performance data provided into the Grid Organizational Memory (GOM) developed by WP4.

An event infrastructure is provided by the monitoring infrastructure. Any components within the workpackage, external services and clients can publish and subscribe events through the event infrastructure.
4 EVENTS, PERFORMANCE AND DEPENDABILITY METRICS OF WORKFLOWS

WP3 services have to capture and provide several performance and dependability metrics that characterize Grid workflow-based applications. The metrics can be used to

- Optimize the execution time of the workflows,
- Validate the functionalities of the workflows (e.g. activity call graphs, coordination constraints),
- Analyze the dependability of the workflows,
- etc.

Performance metrics are built from performance measurements and events (e.g., when an activity is started and when its execution finishes, a failure of a computational node, etc.) obtained. Examples of performance metrics are load balance, computation/communication ratio, service requests per activity, end to end response time, etc. We study pilot applications in order to provide a set of performance metrics that characterize workflow applications. We categorize performance metrics into three levels of abstraction including: code region, activity and workflow region and workflow.

Table 4.1 presents proposed performance metrics at code region level. Performance metrics are divided into sub-categories including execution time and counter.

<table>
<thead>
<tr>
<th>Category</th>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time</td>
<td>ElapsedTime</td>
<td>Elapsed time of the code region.</td>
</tr>
<tr>
<td></td>
<td>UserCPUTime</td>
<td>CPU time spent on user mode</td>
</tr>
<tr>
<td></td>
<td>SystemCPUTime</td>
<td>CPU time spent on system mode</td>
</tr>
<tr>
<td></td>
<td>CommTime</td>
<td>Communication time.</td>
</tr>
<tr>
<td>Counter</td>
<td>NCalls</td>
<td>Number of executions of the code region.</td>
</tr>
<tr>
<td></td>
<td>NSubs</td>
<td>Number of executions of sub regions of the code region.</td>
</tr>
<tr>
<td></td>
<td>TotalTransSize</td>
<td>Size of total data transferred (send and receive).</td>
</tr>
</tbody>
</table>

Table 4.1: Performance metrics at code region level provided by the Monitoring and Instrumentation Service.

Table 4.2 presents proposed performance metrics at activity level. Performance metrics are divided into sub-categories including execution time and counter.

<table>
<thead>
<tr>
<th>Category</th>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time</td>
<td>ElapsedTime</td>
<td>End-to-end response time of an activity.</td>
</tr>
<tr>
<td></td>
<td>ProcessingTime</td>
<td>Time an activity spent in processing state.</td>
</tr>
<tr>
<td></td>
<td>QueuingTime</td>
<td>Time an activity spent in queuing system.</td>
</tr>
<tr>
<td></td>
<td>SuspendingTime</td>
<td>Time an activity spent in suspended state.</td>
</tr>
<tr>
<td>Counter</td>
<td>NCalls</td>
<td>Number of invocations of an activity.</td>
</tr>
<tr>
<td></td>
<td>InTransSize</td>
<td>Size of total data transfered to an activity.</td>
</tr>
<tr>
<td></td>
<td>OutTransSize</td>
<td>Size of total data an activity transfers to another.</td>
</tr>
</tbody>
</table>

Table 4.2: Performance metrics at activity level that are provided by the Performance Analysis Service.

Table 4.3 presents proposed performance metrics at workflow and workflow construct level. Performance metrics are divided into sub-categories including execution time, ratio, synchronization, load balancing, and performance improvement.
Performance metrics at activity level and at workflow and workflow region levels are derived not only from metrics presented in Table 4.1 but also from events containing activity execution status and workflow graphs. Therefore, it is essential that the workflow monitoring provides necessary monitoring data of execution status and the workflow graphs include enough static information about activities.

In order to provide these metrics, the Performance Analysis Service also needs static information about workflows (e.g., workflow graphs) and dynamic information about the calling behavior between monitored resources (e.g., the dependency between two activity instances). The metrics, proposed in the above tables, are neither complete nor fixed. We will experience existing workflows in K-WfGrid and study, extend and revise these metrics accordingly. In addition to the proposed metrics, we will study performance metrics that characterize the dependency between workflows and the resources on which the workflows are executed. Moreover, performance properties and problems will be defined based on performance metrics.

Performance metrics are described in an ontology named WfMetricOnto [48]; WfMetricOnto is based on OWL [35]. Figure 4.1 presents the class describing a metric in WfMetricOnto. The following example presents an OWL description for a metric named ElapsedTime:

```xml
<WfMetric rdf:ID="ElapsedTime">
  <inLevel rdf:datatype="http://www.w3.org/2001/XMLSchema#string">All</inLevel>
  <hasUnit rdf:datatype="http://www.w3.org/2001/XMLSchema#string">us</hasUnit>
</WfMetric>
```
We develop a novel ontology named WfPerfOnto [48] for describing concepts associated with Grid workflows and performance data of Grid workflows. WfPerfOnto addresses essential elements related to the execution and performance of a workflow. It includes basic concepts to describe performance data of workflows such as activity, dependency, performance metric, invoked application, etc. We will describe performance data of workflows by using WfPerfOnto. Moreover, concepts of WfPerfOnto will be used in performance analysis requests. Details of WfMetricOnto and WfPerfOnto can be found in [48].

Besides performance metrics, we also study dependability metrics that can be used to evaluate the reliability, availability and security of K-WfGrid services, infrastructure and workflow applications. Table 4.4 tabulates several dependability metrics that are being investigated.

<table>
<thead>
<tr>
<th>Category</th>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>NSystemFailed</td>
<td>Number of failed invocations of a workflow activity due to a system failure.</td>
</tr>
<tr>
<td></td>
<td>NAppFailed</td>
<td>Number of failed invocations of a workflow activity due to an application failure.</td>
</tr>
<tr>
<td></td>
<td>Downtime</td>
<td>Time intervals during which middleware services and computational nodes are unavailable.</td>
</tr>
<tr>
<td></td>
<td>Uptime</td>
<td>Time intervals during which middleware services and computational nodes are available.</td>
</tr>
<tr>
<td>Availability</td>
<td>Avail</td>
<td>Availability of middleware services and computational nodes; reachability of network paths;</td>
</tr>
<tr>
<td>Security</td>
<td>AuthSucc/AuthzSucc</td>
<td>Authentication/authorization is successful.</td>
</tr>
<tr>
<td></td>
<td>AuthFail/AuthzFail</td>
<td>Authentication/authorization is failed.</td>
</tr>
</tbody>
</table>

Table 4.4: Dependability metrics.
5 GRID MONITORING AND INSTRUMENTATION SERVICE

5.1 ARCHITECTURE

WP3 provides a generic monitoring framework. The role of the framework is to provide access to a diversity of monitoring data, including data related to Grid applications and infrastructure, through a uniform interface.

![WP3 monitoring framework](image)

The MIS components fall into two categories: Domain MIS (D-MIS) and “ordinary” MIS which form a super-peer architecture. This hierarchy is introduced for scalability reasons. Clients (e.g., Performance Analysis Service) access monitoring data by connecting to a D-MIS. The components of the framework use GOM as an information service.

Monitoring and Instrumentation Services expose a web service interface for external clients, while they use a fast interface for low-level intercommunication. The low-level communication interface will be based on existing technologies such as ICE [5]. In order to support both data query and subscription, as well as to enable access to historic monitoring data, collected monitoring data, provided by sensors, will be stored in a performance data...
repository. This repository (e.g. embedded database) must be tightly coupled with the monitoring service in order to provide a performance oriented and scalable monitoring system.

The *sensors* are arbitrary data providers. To comply with the monitoring framework, a sensor must register with a MIS and publish provided data. Note that a sensor can be merely a top-level access point to an underlying complex infrastructure that actually extracts the data. For example, we could use existing distributed systems for monitoring of infrastructure or applications and build sensors on top of those infrastructures. In this case, the sensor would be capable to properly register with a MIS, and expose proper interfaces, while on the other hand it would communicate with the underlying system in its native protocol. In this way we plan to adapt existing systems for monitoring of applications and infrastructure (see sections 5.5 and 5.6). Sections 5.2 and 5.3 describe more detail about sensors.

The *client* can connect to any D-MIS to request for interesting data. The framework will find the data provider which may require to directly query or enable appropriate sensors. The requested data is then returned to the client if the data is available.

*Grid Organizational Memory* (GOM), developed by WP4, is utilized as a knowledge repository and information service. We will use GOM, among others, to support start up of a new MIS. When a MIS connects to the framework, it should know all other monitoring services and information about data provided by those MISs. To enable this, each MIS registers with GOM and publishes information about itself into GOM. On start-up, a MIS can obtain a list of existing MISs from GOM. Section 7.3 will discuss how to publish information about services and data provided to GOM.

![Figure 5.2: Detailed architecture of the Monitoring and Instrumentation Service](image)

Fig. 5.2 depicts a detailed architecture of a single Monitoring and Instrumentation Service. A MIS is composed of five units/modules.

- *Instrumentor* receives *instrumentation requests* and invokes an instrumentation service to conduct the instrumentation according to the requests. The instrumentation service inserts/enables application sensors into the workflow applications; the sensors then collect performance measurements.
- **Data Collector** collects data from a variety of sensor types, e.g., application sensors, event sensors, job monitoring sensors, and computational node monitoring sensors. The role of the sensors is to conduct the measurements, for instance extracting actual data from the monitored resources.

- **Sensor Controller** processes requests controlling the monitoring activities and enables or disables the sensors accordingly. Data collected from sensors is temporarily stored in a Data Repository.

- **Data Provider** processes requests of data **query** and **subscription**. It may access the Data Repository in order to fulfill a request.

- **Data Repository** stores data collected from sensors in a database. The stored data can be retrieved by Data Provider.

The MIS offers query and subscription mechanisms for other services to obtain data. It provides interfaces for querying and subscribing data based on query and subscription language (presented in Section 7.4). A MIS can also collect performance measurements from other MISs. Thus they constitute a peer network of MISs.

---

Figure 5.3: data subscription scenario
Figure 5.3 presents a data subscription scenario which explains the operation of the framework. The client first refers to GOM to obtain the reference to a D-MIS (steps 1, 2 and 3). Next, the client sends a subscribeData request to the D-MIS (step 4). The D-MIS refers to its internal registry to find out who provides the data requested by the client. In this scenario, it is found out that the data is provided by another monitoring service to which the request is forwarded (step 8). The MIS sends the request to its Data Provider component which in this case finds the sensor providing the requested data. The request is forwarded to the sensor (steps 13, 14) which creates a Publisher communication channel and returns its reference all the way back through the MIS and D-MIS to the client. Up to now all invocations in the system were synchronous. Now the client connects directly to the sensor (step 17) through an IceStorm channel (designed for publish-subscribe communication) where data may asynchronously be collected.

5.2 GENERIC SENSOR LIBRARY

The monitoring system can collect and store multiple types of monitoring and performance data provided by a rich set of diverse sensors. To this end, WP3 provides a generic sensor library, based on which many different types of sensors can be developed.

![Generic Sensor Diagram]

Fig. 5.4 illustrates the concept of the generic sensor library. This library can be used by any client, tool and service in order to provide monitoring data to the monitoring system. The client/tool/service needs to

- incorporate the generic sensor library into the entity that produces the data (an application, scheduling system, workflow execution engine, etc.).
- specify a sensor description containing information of provided metrics and events (for example “CPU usage for host X”, “Calls to MPI functions”, etc.) This description will be expressed in an XML-based language.
- implement the code which actually extracts the data (for example reads the current CPU usage via OS mechanisms).

5.3 SENSORS

A sensor, once deployed, discovers a monitoring service, and connects to and registers with the monitoring service. Sensors register to a monitoring service with a sensor description of provided data. There are two types of sensors supported:
- Event-driven: this type of sensors collects and produces data when an event occurs.
- Demand-driven: this type of sensors collects and produces data only when requested (on demand).

Sensors will have customizable feature such as buffering policy and data update rate. Moreover, we consider developing sensors that support rule-based monitoring, for example sensor will provide data only if a threshold is reached. Sensors should also provide functionality by which the monitoring service can control the action of the sensor (e.g. enable/disable the sensor). This work will extend previous work of event-driven, demand-driven and rule-based monitoring sensors [47].

5.4 INTERACTION WITH CLIENTS

Each monitoring service must provide service interfaces through a single access point for other services to (a) control the instrumentation and monitoring, and to (b) obtain monitoring data. Thus, there are three types of service operations:

- Instrumentation Operations: to support the dynamic instrumentation, adding, changing and removing instrumentation at runtime.
- Control Monitoring Operations: to support the control of the monitoring, e.g. to start or to stop a sensor monitoring activity.
- Data Query and Subscription Operations: to process data query and subscription requests from consumers and to provide requested data to consumers.

5.5 APPLICATION MONITORING

The application monitoring system consists of (i) an instrumentation service to control the monitoring activities and (ii) application sensors to collect monitoring data and to deliver it to the monitoring service.

The instrumentation service is used to conduct dynamically enabled instrumentation of Grid workflow applications. It processes requests sent by instrumentation requestors and conducts the instrumentation. The instrumentation is dynamically activated – it can be enabled or disabled at runtime. There are two levels of instrumentation, and consequently, application monitoring:

- **Activity-level monitoring.** Workflow activities are instrumented to measure code regions with selected performance metrics.
- **Workflow-level monitoring.** The generated workflow is instrumented. Therefore, when the workflow is executed we can monitor activities and their status.

5.5.1 Activity-level Monitoring

In general, we should consider two layers at which activities can be monitored: (1) the service layer represents (2) the legacy code layer. The service layer includes external web-service methods exposed by the activity and invoked during the execution of a workflow. An invocation of a service method may involve the execution of legacy code which actually performs the computations. The legacy code can be implemented in programming languages and models which differ from those used for the Web service. For example, the legacy code is based on C/Fortran MPI while the Web service is based on Java.

We aim to support monitoring at both described levels. To achieve that aim, we need to instrument both the service layer and the legacy code layer. Instrumentation of multilingual applications has been described in Section 1.2.
Service-level Monitoring

Monitoring at the service level will be conducted by means of instrumentation of Java code, either at the container level or at the service code level, whichever option proves more viable. The instrumentation will generate events which will be collected by dedicated sensors connected to the monitoring framework.

Legacy-level Monitoring

To monitor the legacy code invoked within Web services, we will adapt existing monitoring systems to work as sensors. An example of such a monitoring system is the OCM-G [16] which supports multi-site parallel applications. The OCM-G is based on the OMIS specification [32] and supports dynamically-enabled instrumentation based on binary wrapping.

To adapt the OCM-G to work with the generic monitoring infrastructure, we will provide an adaptation layer which works as a sensor for the WP3 monitoring framework and a tool for the OCM-G at once. Figure 5.5 presents this concept.

![Figure 5.5: Adaptation of OCM-G to monitoring framework through application sensors](image)

From the viewpoint of the OCM-G, the sensor is a tool connected to the OCM-G which sends monitoring requests expressed in OMIS. For monitoring service, however, this component works as a sensor compliant to the general sensor infrastructure: it provides description of itself to the monitoring service, exposes standard sensor operations, and other functionality provided by generic sensor library.

Thus, the main task of this high-level application sensor is to process queries from monitoring service and convert them to appropriate monitoring requests supported by OCM-G which should result in obtaining the expected metrics.
Supported Metrics

Here we focus on metrics for code regions inside activities. These metrics, obtained from the OCM-G, include time metrics, counter metrics, etc., as proposed in Section 4.

5.5.2 Workflow-level monitoring

In order to support workflow monitoring we will exploit the Java-internal event system of the Grid Workflow Execution Service (GWES). During the invocation of workflow activities the GWES receives monitoring events from the low-level Grid middleware, e.g., using the GRAM protocol [11]. The GWES and its workflow event system will be deployed as a WSRF-compatible Web Service and will provide an interface for publishing workflow-related events. We will develop sensors that can be connected to the GWES. These sensors will provide to the monitoring framework the events and measurements which are required to support the metrics described in Table 4.3. The data collected by these sensors will include information, such as:

- Workflow execution request received
- Instantiating workflow / instantiation finished (success / error)
- Workflow execution state changed (e.g., initiated, running, active, terminated, suspended, completed)
- Transition state changed (e.g., inactive, active, suspended, completed, failed)
- Place state changed (marked, unmarked)
- Refinement requested from User / request answered
- Refinement requested from WCT (Workflow Composition Tool) / request answered
- Refinement requested from AAB (Automatic Application Builder) / request answered
- Refinement requested from Scheduler / request answered
- Instantiating refinement / Instantiation finished (success / error)
- Web service operation invoked / terminated (cause: e.g., finished successfully, error)

All events will be annotated with a timestamp in order to make performance measurements.

5.6 GRID INFRASTRUCTURE MONITORING

In order to collect data such as CPU usage or network bandwidth from particular hosts efficiently we will reuse existing infrastructure monitor systems and adapt them to work in the generic monitoring framework (i.e., as a sensor) in a similar way as described in Section 5.1 about the OCM-G application monitoring system. Currently we plan to use Ganglia [4] for most metrics.

The preliminary list of metrics that are planned to be supported are summarized in Table 5.1.
### Table 5.1: Static and dynamic metrics of Grid infrastructure

<table>
<thead>
<tr>
<th>Category</th>
<th>Metric Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static information</td>
<td>HostName</td>
<td>Machine name</td>
</tr>
<tr>
<td></td>
<td>IPAddr</td>
<td>IP address</td>
</tr>
<tr>
<td></td>
<td>OpSys</td>
<td>Operating system</td>
</tr>
<tr>
<td></td>
<td>CPUType</td>
<td>CPU Type</td>
</tr>
<tr>
<td></td>
<td>MaxMainMemSize</td>
<td>Maximum main memory size</td>
</tr>
<tr>
<td></td>
<td>MaxDiskSize</td>
<td>Maximum disk size</td>
</tr>
<tr>
<td>Dynamic information</td>
<td>MemUsage</td>
<td>Memory usage</td>
</tr>
<tr>
<td></td>
<td>DiskUsage</td>
<td>Disk Usage</td>
</tr>
<tr>
<td></td>
<td>CPUUsage</td>
<td>CPU usage (user/system/idle)</td>
</tr>
<tr>
<td></td>
<td>Avail</td>
<td>Availability</td>
</tr>
<tr>
<td></td>
<td>NetPathBandwidth</td>
<td>Network path bandwidth</td>
</tr>
<tr>
<td></td>
<td>NetPathLatency</td>
<td>Network path latency</td>
</tr>
<tr>
<td></td>
<td>NetPathAvail</td>
<td>Network path availability</td>
</tr>
</tbody>
</table>

**Figure 5.6: Event infrastructure**

## 5.7 EVENT INFRASTRUCTURE

The monitoring service will provide an event infrastructure for K-WfGrid which supports the definition of events, e.g. events related to system problems, application failures, application activities, etc. The monitoring service must maintain book-keeping information about event subscriptions. The monitoring service is responsible for reporting collected events to proper clients.

Focusing on scalability and flexibility, a decentralized P2P-based event system is desirable. Figure 5.6 shows possible elements of an event infrastructure:

- A set of sensors which capture system and user-defined events. The sensors publish their events by means of generic sensor libraries.
- Event delivery system that is based on the monitoring framework
- Event subscribing and querying library for event consumers
- Event notification

Events are transferred from the event source to the event sink by means of a network of decentralized monitoring services.

Recently, in the WSRF-compatible approaches the so-called WS-Notification standard has been established and partially implemented within the Globus Toolkit 4 (GT4) (see Figure 5.7).
WS-Notification is a family of related white papers and specifications that define a standard Web services approach for notification using a topic-based publish/subscribe pattern [26]. It includes:

- Standard message exchanges to be implemented by service providers that wish to participate in Notifications
- Standard message exchanges for a notification broker service provider (allowing publication of messages from entities that are not themselves service providers)
- Operational requirements expected of service providers and requestors that participate in notifications
- An XML model that describes topics.

The WS-Notification family contains three normative specifications: WS-BaseNotification, WS-BrokeredNotification, and WS-Topics. In the GT4, only WS-BaseNotification and WS-Topics is implemented.

The WS-Base Notification specification defines the Web services interfaces for NotificationProducers and NotificationConsumers. It includes standard message exchanges to be implemented by service providers that wish to act as NotificationProducers and NotificationConsumers along with operational requirements expected from the service providers [25].

The WS-Topics specification defines a mechanism to organize and categorize items of interest for subscription known as "topics". These are used in conjunction with the notification mechanisms defined in WS-BaseNotification. WS-Topics defines three topic expression dialects that can be used as subscription expressions in subscribe request messages and other parts of the WS-Notification system. It further specifies an XML mode for describing metadata associated with topics [24].

We will provide a global event infrastructure to enable subscription to events of interest. The core of the event infrastructure will act as a notification broker while the library for publishing and subscribing events used by event producers and consumers will expose WS-Notification interfaces. This event infrastructure will initially be based on the standard Globus Toolkit 4 implementation of WS-Notification. If tests will show huge performance problems with this solution for the use in the monitoring system, other technology, such as ICE will be used.

### 5.8 PERFORMANCE DATA REPOSITORY

In order to support both data query and subscription, as well as access to historic information, performance data needs to be stored. A performance data repository is used to store performance measurements.
The monitoring system should respond to requests as fast as possible in a scalable fashion. Therefore, we plan to integrate a repository with the monitoring service. An embedded database will be used in order to avoid client-server communication with a database.

Since our aim is to exploit profiling data, which contains summary statistics, rather than full trace information, the resulting database size is expected to be relatively small which will perhaps allow us to keep most data in memory.
6 GRID PERFORMANCE ANALYSIS SERVICE

The Grid Performance Analysis Service (PAS) will analyze performance measurements and monitoring data obtained from MIS. PAS supports various aspects of Grid performance analysis:

- Distributed Performance Analysis Service
- Performance Interpretation and Bottleneck Search
- Performance Experiment Repository
- Performance Visualization

6.1 DISTRIBUTED PERFORMANCE ANALYSIS SERVICE (DIPAS)

Although most performance monitoring tools in the Grid operate in a distributed fashion, performance analysis is still implemented as a centralized module. Most performance analysis services obtain performance and monitoring data from distributed sources and analyse this data at a centralized location. As part of this subtask we will design a distributed analysis architecture based on which a distributed performance analysis service (DIPAS) is built. The architecture provides a set of distributed Grid services which collect performance and monitoring data from the monitoring service, collaborate in doing the analysis in a distributed fashion, and provide performance results to various clients. To achieve this objective, we will build Grid services and agents as part of a distributed analysis framework.

Figure 6.1 presents the distributed analysis framework. In our framework we will develop a set of distributed agents. Agents are organized in societies and in every society an agent will be elected as a major. If a major fails then a new agent will be elected for that role. The main role of a major is to communicate with the majors of the other societies. A major also acts as a job coordinator for other agents of its society. Agents within a society communicate in a peer to peer model. Agents communicate with each other by exchanging standard messages whose ontology is described by the WfPerfOnto ontology [48]. Clients of DIPAS will utilize the performance analysis service by invoking service operations provided by major agents. Information of major agents will be published into GOM. Thus any client that wants to request for performance analysis information can discover the analysis service by accessing information published in GOM. A client that wants to send an analysis request should first locate a major by contacting GOM. The client then sends the request to the major which will route the request to the proper society that can answer this request.

Agents obtain monitoring data from MIS and they can control the instrumentation of workflows. If in a society one agent needs the support of another agent to properly answer a request, then the first agent will contact the other agent through a specific protocol. Once an agent has done some analyses it stores performance results into a performance experiment repository. All agents analyzing data for the same workflow reside in the same society.

DIPAS controls the monitoring and instrumentation process, conducts performance analysis for workflows at runtime, and provides performance metrics proposed by a metric ontology. Currently we are investigating a peer-to-peer model for developing distributed analysis agents. Grid services are developed based on GT. Furthermore, agents written in Java are easily integrated into the K-WfGrid framework.

The performance of workflows will be analyzed at various levels of detail including code region, activity, and workflow regions and workflow. A workflow will be divided into workflow regions. We will collect and store different workflow graphs and performance results into a performance experiment repository. Each workflow is stored with associated performance metrics. Based on the performance experiment repository we can conduct multi-experiment analysis. Moreover, because the monitoring service only monitors concrete workflow, the performance analysis has to analyze the performance of concrete workflows and to map the performance results of concrete workflows to abstract workflows.
6.2 PERFORMANCE INTERPRETATION AND BOTTLENECK SEARCH

Most Grid performance analysis approaches provide various kinds of performance data but leave the complex and error-prone interpretation and analysis of such data to the clients. In the context of the K-WfGrid project we want to go beyond current state of the art by

- defining new performance properties for workflow applications, and
- developing a novel bottleneck search module for workflow applications at activity level.

We define new performance properties for workflow applications, including inefficiency, load imbalance, communication, synchronization, loss of parallelism, Grid middleware overhead, total overhead, and unidentified overhead. We will describe the severity of these properties for workflow executions. Performance properties [21] are normalized values in the range $[0, 1]$ that characterize the negative performance behavior and are defined based on performance metrics proposed in Section 4 and provided by MIS and DIPAS. We will study the suitability of the Apart Specification Language [21] and the Java Property Specification Language [41] to express these properties. The performance properties will be organized in a hierarchy to avoid unnecessary evaluations during the search for performance properties.

The systematic search for performance problems is carried-out in the context of an optimization framework, based on our previous experience on developing the ZENTURIO experiment management tool [37, 38]. The problem search space is expressed through the ZEN language [36] that defines arbitrary application parameters through directive-based annotations of arbitrary files.

The parameter space consists of several dimensions that will be studied in this work:

- the performance properties,
• the workflow regions (i.e., sub-workflows of the workflow), and
• the problem size.

Our approach exclusively targets performance analysis of the workflow application at the level of the Workflow Management System (the workflow coordination and execution engine). Activities are considered as black-boxes and are not analyzed internally. The search on the workflow region axis will start from evaluating the properties from the innermost workflow regions and will be expanded to the outermost regions of a single workflow. The collection of workflow monitoring data will be crucial to support the search for performance problems. The static workflow instrumentation (at the Workflow Management System level) will be considered as a simplified starting point. The problem size axis requires different workflow executions. The schedule axis will attempt to find workflow mappings that minimize the performance properties. The search engine tool will declare as bottleneck any property that cannot be minimized below a user-defined severity. For the realization of the search engine on the schedule axis we will investigate a variety of experimental heuristics, including linear programming, gradient descent methods, evolutionary algorithms, etc.

![Diagram](image.png)

**Figure 6.2: Performance bottleneck analysis for workflows.**

### 6.3 PERFORMANCE EXPERIMENT REPOSITORY

The performance analysis service will store performance results associated with workflows into a performance experiment repository. This repository is also used to store performance properties, bottleneck conditions, etc., that are used or provided by the performance interpretation and bottleneck search.

Note that the experiment repository is not used to store monitoring data (which is expressed in XML and stored by the performance data repository, see Section ). Instead it is used to store summarized information about experiments and workflows. Such information can be described in WfPerfOnto.

GOM is one option to store performance results (see Section 8.2). At this stage of the project, we are unsure whether GOM can fulfill necessary DIPAS performance constraints (time to access GOM and transfer data to/from GOM). We will collaborate with GOM developers in determining whether performance results can be stored into GOM. Meanwhile, we plan to use PostgreSQL [8] for our performance experiment repository.¹

### 6.4 PERFORMANCE VISUALIZATION

A graphical user interface (GUI) is developed to visualize performance information at various levels of detail, for example, workflow structure, workflow activities, Grid site, and network. A visualization and user portal system

¹To store OWL-based ontological data we plan to use Jena [27]. The back-end database that Jena supports can be PostgreSQL. Also we need to store bottleneck conditions, which are not described in XML or OWL. Therefore, PostgreSQL would be the best solution for the experiment data repository at the time of writing this report.
WP3: Appendix to the Deliverable D2.5.1

is developed to provide a mechanism to control and observe performance instrumentation, monitoring, and event and analysis infrastructure. This portal allows the user to write instrumentation and performance data query and subscription requests, and to specify events. Moreover, the visualization system supports the display of system faults and application faults, and performance degradation.

The visualization for workflow performance analysis should integrate

- **Workflow Execution Monitoring**: visualize the execution status of workflows activities. Activity events and states are associated with each activity.

- **Instrumentation GUI**: support the instrumentation of workflows. The visualization will display the workflow structure, active activities, and allow the user to select active activities and lists of instrumented functions within invoked applications of active activities, to specify instrumentation requests and to conduct the instrumentation.

- **Subscription and Query Performance Data**: support the user to specify and conduct data query and subscription requests.

- **Workflow Performance Analysis**: visualize workflows together with their associated performance metrics. This visualization should provide multiple levels of view, e.g. workflow, workflow region, activity, etc.

In order to visualize workflows, we plan to use the workflow visualization library developed by WP2. ASKALON Visualization Diagram [14] and JFreeChart [28] will be used for visualizing performance data. In addition, we will reuse libraries that WP2 uses to manipulate workflow structures.
7 PERFORMANCE SERVICE INTERFACES AND DATA REPRESENTATIONS

7.1 STANDARDIZED INTERMEDIATE REPRESENTATION FOR INVOKED APPLICATIONS

Instrumentation of applications is the most popular method for gathering performance information required by application performance analysis; the application can be statically instrumented or dynamically instrumented before or during the execution. Normally instrumentation engines for different programming languages and for different instrumentation strategies provide different representations and interfaces for performance tools to control the instrumentation. That hinders a performance analysis tool to interface with multiple underlying monitoring and instrumentation services, especially when conducting the instrumentation of multilingual applications. The APART working group has proposed a Standardized Intermediate Representation (SIR) as an abstract program representation for procedural and object-oriented programs [43]. Basically a SIR contains information about statement and directive types with very little details on the structure of individual statements and directives but it provides enough information about the application for high-level tools to make a decision about code regions that should be instrumented.

SIR is an XML-based representation that includes information about program units (e.g. functions, methods), code regions (e.g. function calls, loops, statements), etc. SIR is designed for describing Fortran, Java, C and C++ in source code format. Therefore, it supports a rich set of information which is available when parsing a source code programs. However, with our dynamic-enabled instrumentation, in which the intended instrumented program are pre-instrumented and instrumented code can be enabled or disabled at the runtime, the information obtained is substantially reduced. Instrumented code in WP3 mostly limits to program units and function calls. Previously, we developed SIRBC (SIR for Binary Code) [46] to represent SIR of binary code. SIRBC represents binary code at level of program units and function calls. Therefore, in WP3, we will extend SIRBC to represent code regions of invoked applications to be instrumented.

SIRWF (SIR for Workflow) extends SIRBC by adding information about the workflow, activity and invoked application. Similar to SIRBC, SIRWF currently supports only level of program unit and function call. An application process is represented as a set of program units. Each program unit contains a set of code regions. A code region is a function call that contains information (e.g. name, source information) about the calling function. Each program unit or code region is associated with a unique identifier. The requester uses that identifier to specify a program unit or a code region. By using SIRBC, the instrumentation requester can understand the application structure and make instrumentation requests specifying code regions that should be instrumented. Initial schema of SIRWF can be found in Appendix B.

SIRWF is used to describe the structure of invoked applications of workflow activities. Based on SIRWF, the instrumentation requesters can select interesting code regions for instrumentation.

7.2 WORKFLOW INSTRUMENTATION REQUEST LANGUAGE

The workflow instrumentation request language (WIRL) will be developed and used as the language between the instrumentation requester, e.g. the Performance Analysis Service, and instrumentation service. WIRL is based on IRL [46] , and MIR [42]. WIRL is a request and response protocol: WIRL requests specify instrumentation requests whereas WIRL responses return the result of the instrumentation requests.

A WIRL request consists of experiment information and instrumentation tasks. Experiment information identifies applications to be instrumented. Experiment information, obtained from the MS, includes
- **Workflow instance identifier**: identifies the workflow instance.
- **Activity identifier**: identifies the workflow activity.
- **Invoked application identifier**: identifies the invoked application of the activity.
- **Computational node**: identifies the computational node on which the invoked application is executed.
- **Process identifier**: identifies the application process of an invoked application of an activity.
- **Thread identifier**: identifies the thread of an application process.

The workflow instance identifier is used to distinguish performance and monitoring data of different experiments. **Instrumentation tasks** specify instrumentation and other operations. WIRL contains the following tasks:

- **attach**: requests the instrumentation service to attach the application and to prepare next operations
- **getsr**: requests the instrumentation service to provide list of instrumented functions within the applications
- **enable/disable**: enables or disables an instrumented code.
- **finalize**: finishes the instrumentation

An instrumentation task may contain the following information:

- **Code region identifiers**: used to determine code regions to be instrumented.
- **Metric names**: used to determine metrics of interest.
- **User-defined events**: used to specify user-defined events that should be generated at given points of the application.

Code region identifiers are obtained from the list of instrumented functions provided by the MS. Metric names are pre-defined; performance metrics are specified by the metric ontology, WfMetricOnto (see Section 4).

The use of code region identifier allows to indicate the code region to be measured. However, in many cases one may want to instrument all the code regions in a program unit or all code regions which have the same name (e.g. the same function but being called several times at different locations). The WIRL and instrumentation service must support this feature.

A **WIRL instrumentation response** contains the name of the request, the status of the request (e.g. OK, FAIL) and possibly a detailed responding information encoded in `<![CDATA[ ... ]]>` tag.

The current version of WIRL can be found in Appendix C.

### 7.3 DATA REPRESENTATION

#### 7.3.1 XML Representation for Performance and Monitoring Data

Performance measurements, monitoring data and events of applications and infrastructures are represented in XML. Each type of performance and monitoring data is provided by a type of sensors. A message containing monitoring data of a monitored resource (e.g. machine, network path, code region) should consist of the following information:

- **Resource identifier**: indicates information about the monitored resource.
- **Data type identifier**: indicates the type of the monitoring data.
- **Experiment information**: indicates the experiment in which the monitoring is conducted. The experiment information may be required only for application monitoring data. It is associated with the monitoring data so that we can determine the experiment with which the monitoring data is collected.

- **Performance measurements**: are performance and monitoring data of a resource that the sensor measures, gathers or collects.

A generic structure of a message, which is used to describe the performance data, is outlined as follows:

```xml
<MonitoringData dataTypeID="datatypoid" resourceID="resourceid">
<!-- XML describes experiment information -->
<!-- XML describes performance measurements -->
....
</MonitoringData>
```

where `datatypoid`, `resourceid`, are data type identifier and resource identifier, respectively. The part expressing performance measurements is dependent on each type of sensors. Based on `dataTypeID`, `resourceID` and experiment information, the monitoring service can prepare buffers and store messages containing performance data to the appropriate buffers. As each type of data is provided by a type of sensors, the sensor identifier may also be used as the identifier of the data type that the sensor provides. Note that an implementation of a sensor may not provide only a single type of data.

Each above-mentioned message is an XML document. The size of an XML document is relatively small but it is possible that the monitoring service has to manage a large amount of documents. Therefore, embedded database, e.g. Berkeley DB [10], will be more suitable for implementing the performance data repository of monitoring service (see Section 5).

Table 7.1 presents proposed XML schemas of performance data. These XML schemas are built based on those implemented in SCALEA-G [46]. Drafts of some schemas can be found in Appendix D.2.

<table>
<thead>
<tr>
<th>Data type ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host.cpu.used</td>
<td>CPU usage of a computational node</td>
</tr>
<tr>
<td>host.mem.used</td>
<td>Memory usage of a computational node</td>
</tr>
<tr>
<td>host.system.loadavg</td>
<td>Load average of a computational node.</td>
</tr>
<tr>
<td>path.bandwidth.achievable.TCP</td>
<td>maximum amount of MB/s that can be transferred through TCP paths</td>
</tr>
<tr>
<td>path.delay.roundtrip</td>
<td>time needed for a packet to travel through an IP path and to go back.</td>
</tr>
<tr>
<td>path.delay.roundtrip.TCP</td>
<td>time taken for a packet to travel through a TCP path and to go back.</td>
</tr>
<tr>
<td>app.prof</td>
<td>Profiling data of applications.</td>
</tr>
<tr>
<td>app.event</td>
<td>Application event.</td>
</tr>
<tr>
<td>wf.event</td>
<td>Execution status of workflow activities.</td>
</tr>
</tbody>
</table>

Table 7.1: XML schemas of performance data.

### 7.3.2 OWL Descriptions of Monitoring Data

Information about supported and available monitoring data has to be published so that clients can access and retrieve interesting monitoring data. We will publish information about provided monitoring data into GOM developed in WP4. Information about monitoring data will be described by OWL. OWL descriptions of information about monitoring data will be developed in parallel with the development of XML schemas for monitoring data.

Figure 7.1 presents a draft of an OWL description of information about monitoring data. The class `MonitoringData` describes the concept of information about monitoring data. `MonitoringData` has four properties:

- **Property hasDataType**: specifies the sensor which generates the monitoring data.
• Property ofResource: specifies the resource to be monitored.
• Property isStoredIn: specifies the handle of the Monitoring Service which provides the monitoring data.
• Properties validFrom, validTo: specify the duration in which the data is available.

![Diagram](image)

Figure 7.1: Description of OWL/RDF information about monitoring data

An example of OWL description for host.cpu.used is presented as follows:

```xml
<MonitoringData rdf:ID="host.cpu.used">
  <validFrom rdf:datatype="http://www.w3.org/2001/XMLSchema#long" >12345678</validFrom>
  <isStoredIn rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI" >http://bridge.vcpc.univie.ac.at:8765/ogsa/services/MonitoringService</isStoredIn>
  <ofResource rdf:datatype="http://www.w3.org/2001/XMLSchema#string" >shareck.dpsuibk.ac.at</ofResource>
  <validTo rdf:datatype="http://www.w3.org/2001/XMLSchema#long" >12345687</validTo>
  <hasDataType rdf:datatype="http://www.w3.org/2001/XMLSchema#string" >host.cpu.used</hasDataType>
</MonitoringData>
```

Further information about monitoring data, e.g. whether data can be queried and/or subscribed or can only be queried, will be studied and described. From information about monitoring data, a client knows which WP3 service it should contact in order to obtain required data.

Note that although descriptions about monitoring data are represented in OWL, the monitoring data is described in XML representation and the data query and subscription requests will be based on XML (see Section 7.4). Therefore, the clients have to specify the XML-based requests by using OWL-based descriptions.

### 7.3.3 OWL Descriptions of Monitoring and Performance Analysis Service

Information about WP3 services (e.g. service name, service operation, lifetime, etc) has to be published so that other clients and services can discover WP3 services and utilize them. Information about MIS and PAS will be expressed in OWL/OWL-S and published into GOM. This task will be done in parallel with the development of service interfaces for WP3 services.

Although GOM provides facilities for searching its registry service. It is unknown yet how the client discovers WP3 services and performance data provided by WP3. Therefore, it is possible that WP3 has to provide a simple library for the client to find WP3 information. This library will utilize search facilities provided by GOM and provide the client with information about WP3.
7.4 PERFORMANCE DATA QUERY AND SUBSCRIPTION REQUEST

WP3 services support data query and subscription, and notification. Requests for data query and subscription will be expressed in a pre-defined XML schema named PDQS (Performance Data Query and Subscription). PDQS requests will be used in service interfaces for data query and subscription. A draft of PDQS schema can be found in Appendix E.

7.4.1 Data Subscription and Query

Data subscription requests contain the following information:

- **Subscription time**: specifies the duration the subscription is valid.
- **Sensor identifier and resource identifier**: are used to determine the type of performance data.
- **Data filters**: are used to filter the content of performance data.
- **Aggregate operators**: are used to aggregate the resulting data before sending it back to the requester.

The data filters will be expressed in XPath/XQuery [52, 53]. Consider the monitoring and performance data and events in WP3 described in XML and a large set of libraries and native XML databases supporting data query and filter with XPath/XQuery. XPath/XQuery data filters can easily be used to query against an XML document (a message containing performance measurements) by using any of suitable libraries which substantially reduces work done by the monitoring service.

Not all the above-mentioned information in subscription requests are required. For example, the data filter of a request can contain sensor identifier and resource identifier, thus the request does not necessarily include the sensor identifier and resource identifier.

A data query request is similar to a data subscription request except that the data query request is not associated with a subscription time. There are some types of monitoring data which will not be available for subscription mode, but only for query mode. For example, the data provided by a demand-driven sensor can be delivered when a client makes a query to get the data. Based on the query the monitoring service will control the sensor to collect the data; without a query, the data will not be collected.

7.4.2 Notification

There are two types of subscriptions. Normally, the subscription is successful only if the type of data requested is known at the time the subscription is made. However, in many cases the type of requested data is not known until an event that generates the data occurs. In both types, notification mechanism allows a client to get requested data from the monitoring service.

7.4.3 Example of PDQS

PDQS requests will be used in service interfaces for data query and subscription. The following example presents a subscription request of the draft PDQS language.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PDQS>
  <dataTypeID>path.bandwidth.capacity.TCP</dataTypeID>
  <subscriptionTime>
    <from>1110791011475</from><to>1110791111475</to>
  </subscriptionTime>
</PDQS>
```
7.5 SERVICE OPERATIONS

WP3 services will be OGSA-based services. We will rely on GT which provides a reference implementation of Web Service Resource Framework (WSRF) [19] for developing WP3 OGSA services.

Service operations of WP3 Grid services are categorized into

- **Request-reply mode**: service operations are based on request-reply model, for example operations for querying data. This type of service operations supports synchronous/blocking calls. In this model, the client invokes the service operation and gets the result.

- **Callback model**: service operations are based on callback model, for example operations for subscribing data. This type of service operations supports asynchronous/non-blocking calls. In this model, the client invokes the service operation and gets a result identifier. The resulting data will be delivered when the data is available by using callback mechanism.

Moreover, a client of performance monitoring and analysis services may or may not be a web service. Therefore, every WP3 service should provide a library which can be used by clients to utilize the service. Basically, this library contains APIs to access the service.

Any monitoring service or instrumentation service or analysis service should provide some generic service operations beside its specific service operations. Generic service operations support common tasks whereas specific service operations implement specific features of a service.

7.5.1 Generic Service Operations

The full and detailed list generic service operations will be proposed. Here are examples of generic service operations

- **ping**: supports the ping service.

- **serviceLoad**: provides load information of the service, for example number of requests the service has served from the time the service starts, average processing time of a request, etc.

7.5.2 Specific Service Operations

The full and detailed list of specific service operations will be proposed. Here are examples of some specific service operations

- **subscribeData**: supports the data subscription.

- **queryData**: supports the data query.

- **instrument**: supports the instrumentation.

- **perfAnalysis**: supports the analysis
8 WORKPACKAGES DEPENDENCIES

8.1 DEPENDENCIES BETWEEN WP3 AND WP2

WP3 requires data about the workflow and resources on which workflow activities are executed, and execution status of workflow activities. All of these data are provided by WP2. For example, the execution status can be obtained from the GWES (Grid Workflow Execution Service). These data may be static or dynamic. Table 8.1 presents information that WP3 requires from WP2.

<table>
<thead>
<tr>
<th>Types of data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow</td>
<td>Workflow graph representation The graph of a workflow which must include detailed information about workflow structures and invoked applications. WP3 will reuse a WP2 library for traversing and manipulating the workflow graph.</td>
</tr>
<tr>
<td>Invocation and</td>
<td>Names of States and Events</td>
</tr>
<tr>
<td>Control Model</td>
<td>A list of names of states and events names (e.g. submitted, active) of activities. This information is static.</td>
</tr>
<tr>
<td>State Transition</td>
<td>State Transition machine The state transition machine is implemented in WP2.</td>
</tr>
<tr>
<td>machine</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.1: Information required by WP3 to be provided by WP2.

WP3 has to work closely with WP2 in order to obtain events of the execution status of workflows. Execution status contains information about the execution of an activity such as activity name, event name, resources on which invoked applications of activities are executed, OS process identifiers of invoked applications of active activities, etc. WP3 must know activity states, events, and details about the state transition machine in order to implement activity monitoring and analysis.

In order to obtain execution status of workflows, WP3 provides sensor library to WP2. By using the sensor library, WP2 can collect and send execution status to the monitoring service. WP2 and WP3 collaborate in capturing and collecting execution status of activities for performance monitoring.

For each workflow, WP3 uses the workflow graph to analyze the performance of the workflow and to associate performance measurements and results with activities of workflows. Execution status is used to determine the status of activities and the resources on which activities are executed. Based on information about the status and the resources, WP3 service will conduct other tasks such as instrumenting invoked applications of activities, monitoring resources, etc.

Table 8.2 presents the information that WP3 can provide to WP2. WP3 provides a rich set of monitoring data and performance results of workflows, computational resources and networks. WP2 accesses the data and results by using facilities provided by WP3, e.g. data query and subscription.

It is a potential that the WTC (Workflow Composition Tool, WP2) and the AAB (Automatic Application Builder) will need performance and monitoring data (e.g. the capability of a computational node) for constructing workflows.

8.2 DEPENDENCIES BETWEEN WP3 AND WP4

In the current design, WP3 will publish information about WP3 services (Monitoring and Instrumentation Service, Performance Analysis Service) and about provided data (workflows, computational resources, networks) into GOM (Grid Organizational Memory) developed by WP4. GOM should allow any client to access and store information independent from where the data is actually stored.
### Types of data

<table>
<thead>
<tr>
<th>Resources</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Availability of resources</strong></td>
<td></td>
</tr>
<tr>
<td>Computational node</td>
<td>States: available, unavailable</td>
</tr>
<tr>
<td>Network path</td>
<td>States: available, unavailable, high, low, etc.</td>
</tr>
<tr>
<td>Service</td>
<td>States: available, unavailable</td>
</tr>
<tr>
<td><strong>Statistics of the availability of resources</strong></td>
<td>Available and unavailable percentage, reliability measure</td>
</tr>
<tr>
<td><strong>Current status of resources</strong></td>
<td></td>
</tr>
<tr>
<td>Computational node</td>
<td>Memory and CPU usage, Load average, etc.</td>
</tr>
<tr>
<td>Network path</td>
<td>Bandwidth and latency of TCP, IP</td>
</tr>
<tr>
<td><strong>Statistics of status of resources</strong></td>
<td></td>
</tr>
<tr>
<td>Computational node, Network path</td>
<td>AVG, MIN, MAX, etc. Time series (e.g. last hour)</td>
</tr>
<tr>
<td><strong>Status of application executions</strong></td>
<td></td>
</tr>
<tr>
<td>Application processes</td>
<td>States: active, waiting, stopped</td>
</tr>
<tr>
<td>Resource</td>
<td>Resource consumption: CPU usage, memory usage</td>
</tr>
</tbody>
</table>

Table 8.2: Information that WP3 provides WP2.

In order to use GOM, WP3 needs to

- understand data representations in GOM. Meta-data is described in OWL/RDF and service information is described in OWL-S.
- study the scaling and performance behavior of GOM in order to determine whether GOM can be used for performance monitoring and analysis. GOM provides a registry service which supports add, remove, update and search operations. The search can be done through RDQL [35] as well.

Moreover, WP3 considers to put performance results (e.g. workflow and its associated metrics) into GOM if GOM is fast and scalable enough. According to GOM developers, performance results can be stored into GOM if performance results are aggregated values that can be described in small OWL documents. We will work closely with GOM developers in order to determine which performance results can be stored into GOM. The solution could be that full results of performance experiments are stored into the performance experiment repository and only extracted, aggregate performance results are described in WfPerfOnto and stored into GOM.

### 8.3 DEPENDENCIES BETWEEN WP3 AND WP5

The KAA (Knowledge Assimilation Agent, WP5) will collect events provided by WP3 services. We will study events samples from WP3 services that KAA is interested in. The UAA (User Assistant Agent) which provides knowledge about workflows to the end-user can access performance of the whole/part of workflows.
# A ABBREVIATIONS

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAB</td>
<td>Automatic Application Builder</td>
</tr>
<tr>
<td>DIPAS</td>
<td>Distributed Performance Analysis Service</td>
</tr>
<tr>
<td>GOM</td>
<td>Grid Organizational Memory</td>
</tr>
<tr>
<td>GWES</td>
<td>Grid Workflow Execution Service</td>
</tr>
<tr>
<td>IS</td>
<td>Instrumentation Service</td>
</tr>
<tr>
<td>IRL</td>
<td>Instrumentation Request Language</td>
</tr>
<tr>
<td>KAA</td>
<td>Knowledge Assimilation Agent</td>
</tr>
<tr>
<td>MIS</td>
<td>Monitoring and Instrumentation Service</td>
</tr>
<tr>
<td>MS</td>
<td>Monitoring Service</td>
</tr>
<tr>
<td>PAS</td>
<td>Performance Analysis Service</td>
</tr>
<tr>
<td>SIRBC</td>
<td>Standardized Intermediate Representation for Binary Code</td>
</tr>
<tr>
<td>UAA</td>
<td>User Assistant Agent</td>
</tr>
<tr>
<td>WF</td>
<td>Workflow</td>
</tr>
<tr>
<td>WfMS</td>
<td>Workflow Management System</td>
</tr>
<tr>
<td>WIC</td>
<td>Workflow Invocation and Control</td>
</tr>
<tr>
<td>WIRL</td>
<td>Workflow Instrumentation Request Language</td>
</tr>
<tr>
<td>WCT</td>
<td>Workflow Composition Tool</td>
</tr>
<tr>
<td>WP</td>
<td>Workpackage</td>
</tr>
</tbody>
</table>
B SIRWF SCHEMA

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://net.kwfgrid.dr/sirwf"
    xmlns:tns1="http://net.kwfgrid.dr/pu"
    xmlns:tns2="http://net.kwfgrid.dr/cr">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      SIRWF (Standardized Intermediate Representation for Workflow)
      Version 0.1
      Written based on SIRBC (SIR for Binary Code) developed in
      SCALEA-G (http://dps.uibk.ac.at/projectsSCALEA-G)
    </xsd:documentation>
  </xsd:annotation>

  <xsd:import
      namespace="http://kwfgrid.net/dr/pu"
      schemaLocation="pu.xsd" />
  <xsd:import
      namespace="http://kwfgrid.net/dr/cr"
      schemaLocation="cr.xsd" />

  <xsd:element name="sirwf" type="SIRWF"/>
  <xsd:complexType name="SIRWF">
    <xsd:sequence>
      <xsd:element name="sirApp" type="SIRApp" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="SIRApp">
    <xsd:sequence>
      <xsd:element name="experiment" type="SIRWFExperiment" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="sir" type="SIR" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="SIRWFExperiment">
    <xsd:sequence>
      <xsd:element name="wfInstanceID" type="xsd:string"/>
      <xsd:element name="pu" type="tns1:ProcessingUnit" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="SIR">
    <xsd:sequence>
      <xsd:element name="unit" type="SIRUnit" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
<xsd:complexType name="SIRUnit">
  <xsd:sequence>
    <xsd:element name="linestart" type="xsd:integer"/>
    <xsd:element name="lineend" type="xsd:integer"/>
    <xsd:element name="sourcefile" type="xsd:string"/>
    <xsd:element name="coderegion" type="SIRCodeRegion"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="type" type="tns2:CodeRegionType" minOccurs="0" maxOccurs="1"/>
  <xsd:attribute name="id" type="xsd:string" minOccurs="0" maxOccurs="1"/>
</xsd:complexType>

<xsd:complexType name="SIRCodeRegion">
  <xsd:sequence>
    <xsd:element name="linestart" type="xsd:integer"/>
    <xsd:element name="lineend" type="xsd:integer"/>
    <xsd:element name="sourcefile" type="xsd:string"/>
    <xsd:element name="callee" type="SIRCallee" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="type" type="tns2:CodeRegionType"/>
  <xsd:attribute name="id" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="SIRCallee">
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>

</xsd:schema>
C  WORKFLOW INSTRUMENTATION REQUEST LANGUAGE

C.1  WIRL SCHEMA

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://net.kwfgrid/dr/wirl"
    xmlns:tns1="http://net.kwfgrid/dr/pu"
    xmlns:tns2="http://net.kwfgrid/dr/events">

    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            Workflow Instrumentation Request Language (WIRL) schema
            Version 0.1
            Copyright 2004-2005, Hong-Linh Truong. All rights reserved.
            Written based on IRL (Instrumentation Request Language) developed in
            SCALEA-G (http://dpsuibk.ac.at/projects/scaleag)
        </xsd:documentation>
    </xsd:annotation>

    <xsd:import
        namespace="http://kwfgrid.net/dr/pu"
        schemaLocation="pu.xsd" />

    <xsd:import
        namespace="http://kwfgrid.net/dr/events"
        schemaLocation="events.xsd" />

    <xsd:element name="wirl" type="WIRL" />

    <xsd:complexType name="WIRL">
        <xsd:sequence>
            <xsd:element name="experiment" type="WIRLExperiment" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="request" type="WIRLRequest"
                minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="response" type="WIRLResponse"
                minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="WIRLRequest">
        <xsd:sequence>
            <xsd:element name="experiment" type="WIRLExperiment" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="task" type="WIRLTask" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="RequestName"/>
    </xsd:complexType>
```

<xsd:complexType name="WIRLExperiment">
  <xsd:sequence>
    <xsd:element name="wfInstanceID" type="xsd:string"/>
    <xsd:element name="pu" type="tns1:ProcessingUnit" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="WIRLTask">
  <xsd:sequence>
    <xsd:element name="coderegion" type="WIRLCodeRegion" minOccurs="1" maxOccurs="unbounded" />
    <xsd:element name="metrics" type="MetricList" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="events" type="EventList" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="WIRLCodeRegion">
  <xsd:attribute name="unit" type="xsd:string"/>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="id" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="WIRLResponse">
  <xsd:sequence>
    <xsd:element name="detail" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="RequestName"/>
  <xsd:attribute name="status" type="ResponseStatus"/>
</xsd:complexType>

<xsd:simpleType name="MetricList">
  <xsd:list itemType="xsd:string"/>
</xsd:simpleType>

<xsd:complexType name="EventList">
  <xsd:sequence>
    <xsd:element name="event" type="SingleEvent" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="location" type="InstrumentLocation"/>
</xsd:complexType>

<xsd:complexType name="SingleEvent">
  <xsd:sequence>
    <xsd:element name="eventname" type="xsd:string"/>
    <xsd:element name="eventdata" type="tns2:EventAttribute" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="function" type="FunctionList" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="FunctionList">
  <xsd:list itemType="xsd:string"/>
</xsd:simpleType>
C.2 WIRL REQUEST NAMES

Attribute name of element WIRLRequest specifies WIRL request names. Table C.1 presents WIRL request names.

<table>
<thead>
<tr>
<th>Request name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTACH</td>
<td>to attach a given application</td>
</tr>
<tr>
<td>GETSIR</td>
<td>to get instrumented code of the application</td>
</tr>
<tr>
<td>ENABLE</td>
<td>to enable instrumented code</td>
</tr>
<tr>
<td>DISABLE</td>
<td>to disable instrumented code</td>
</tr>
<tr>
<td>FINALIZE</td>
<td>to finish the instrumentation of an application</td>
</tr>
</tbody>
</table>

Table C.1: WIRL instrumentation request names.

C.3 INSTRUMENTATION LOCATIONS

Attribute location of EventList specifies the location where probes generating user-defined events should be inserted. Table C.2 specifies the location of probes generating user-defined events.
### Table C.2: Event instrumentation location.

<table>
<thead>
<tr>
<th>Location Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEFORE</td>
<td>probes generating user-defined events will be inserted before the code region</td>
</tr>
<tr>
<td>AFTER</td>
<td>probes generating user-defined events will be inserted after the code region</td>
</tr>
<tr>
<td>BOTH</td>
<td>probes generating user-defined events will be inserted before and after the code region</td>
</tr>
</tbody>
</table>
D DATA REPRESENTATION

D.1 COMMON SCHEMAS

D.1.1 Profiled Code Regions

```xml
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://net.kwfgrid/dr/cr"
    xmlns:tns="http://net.kwfgrid/dr/cr">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      CodeRegion schema: describing source code information of a code region profiled.
    </xs:documentation>
  </xs:annotation>

  <xs:complexType name="CodeRegion">
    <xs:sequence>
        <!-- information obtained from SIRWF -->
        <xs:element name="name" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="type" type="tns:CodeRegionType" minOccurs="0" maxOccurs="1"/>
        <xs:element name="id" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="startline" type="xs:integer" minOccurs="0" maxOccurs="1"/>
        <xs:element name="endline" type="xs:integer" minOccurs="0" maxOccurs="1"/>
        <xs:element name="filename" type="xs:string" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <!--probeId describes runtime profiler associated with this code region -->
    <xs:attribute name="probeId" type="xs:integer"/>
    <xs:attribute name="probeIdParent" type="xs:integer"/>
  </xs:complexType>

  <xs:simpleType name="CodeRegionType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="call"/>
      <xs:enumeration value="function"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

D.1.2 Processing Unit

```xml
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://net.kwfgrid/dr/pu"
    xmlns:tns="http://net.kwfgrid/dr/pu">\n</xs:schema>
```
D.1.3 Single Event

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://net.kwfgrid/dr/events"
    xmlns:tns="http://net.kwfgrid/dr/events">
    
    <xsd:complexType name="SingleEvent">
        <xsd:sequence>
            <xsd:element name="eventName" type="xsd:string"/>
            <xsd:element name="eventtime" type="xsd:long"/>
            <xsd:element name="eventdata" type="tns:EventAttribute"
                minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="EventAttribute">
        <xsd:sequence>
            <xsd:element name="attrname" type="xsd:string"/>
            <xsd:element name="attrvalue" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```
D.2 XML SCHEMAS OF MONITORING DATA

D.2.1 host.cpu.used

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://net.kwfgrid/dr/hostcpuused">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            Host CPU Usage Schema.
            Version 0.1
            Copyright 2004-2005, Hong-Linh Truong. All rights reserved.
            Written based on Host CPU Usage Schema developed in SCALEA-G
            (http://dps.uibk.ac.at/projects/scaleag)
        </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="monitoringData" type="MonitoringData"/>
    <xsd:complexType name="MonitoringData">
        <xsd:sequence>
            <xsd:element name="hostname" type="xsd:string"/>
            <xsd:element name="eventtime" type="xsd:long"/>  
            <xsd:element name="cpuInfo" type="CPUInfo" minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="dataTypeID" type="xsd:NMTOKEN" fixed="host.cpu.used"/>
        <xsd:attribute name="resourceID" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="CPUInfo">
        <xsd:sequence>
            <xsd:element name="userTime" type="xsd:double"/>
            <xsd:element name="systemTime" type="xsd:double"/>
            <xsd:element name="idleTime" type="xsd:double"/>
            <xsd:element name="waitTime" type="xsd:double"/>
        </xsd:sequence>
        <xsd:attribute name="cpuID" type="xsd:string"/>
    </xsd:complexType>
</xsd:schema>

D.2.2 host.system.loadavg

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://net.kwfgrid/dr/hostsystemload">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            Host system load schema
            Version 0.1
        </xsd:documentation>
    </xsd:annotation>
</xsd:schema>
D.2.3 host.mem.used

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:targetNamespace>http://net.kwfgrid/dr/hostmemused</xsd:targetNamespace>
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            Host memory usage schema
            Version 0.1
            Copyright 2004-2005, Hong-Linh Truong. All rights reserved.
            Written based on host memory usage schema developed in SCALEA-G
            (http://dps.uibk.ac.at/projects/scaleag)
        </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="monitoringData" type="MonitoringData" />
    <xsd:complexType name="MonitoringData">
        <xsd:sequence>
            <xsd:element name="hostname" type="xsd:string"/>
            <xsd:element name="eventtime" type="xsd:long"/>
            <xsd:element name="availmem" type="xsd:long"/>
            <xsd:element name="usedmem" type="xsd:long"/>
            <xsd:element name="totalmem" type="xsd:long"/>
        </xsd:sequence>
        <xsd:attribute name="dataTypeID" type="xsd:NMTOKEN" fixed="host.mem.used"/>
        <xsd:attribute name="resourceID" type="xsd:string"/>
    </xsd:complexType>
</xsd:schema>

Copyright 2004-2005, Hong-Linh Truong. All rights reserved.
Written based on Host system load schema developed in SCALEA-G
(http://dps.uibk.ac.at/projects/scaleag)
D.2.4 app.prof

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://net.kwfgrid/dr/appprof"
  xmlns:tns2="http://net.kwfgrid/dr/pu"
  xmlns:tns3="http://net.kwfgrid/dr/cr">

  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Application Profiling Data Schema
      Version 0.1
      Copyright 2004-2005, Hong-Linh Truong. All rights reserved.
      Written based on application profiling schema developed in SCALEA-G
      (http://dps.uibk.ac.at/projects/scaleag)
    </xsd:documentation>
  </xsd:annotation>

  <xsd:import namespace="http://kwfgrid.net/dr/pu" schemaLocation="pu.xsd"/>
  <xsd:import namespace="http://kwfgrid.net/dr/cr" schemaLocation="cr.xsd"/>

  <xsd:element name="monitoringData" type="MonitoringData"/>
  <xsd:complexType name="MonitoringData">
    <xsd:sequence>
      <xsd:element name="pu" type="tns2:ProcessingUnit" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="cr" type="tns3:CodeRegion" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="metrics" type="MetricList" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="dataTypeID" type="xsd:NMTOKEN" fixed="app.prof"/>
    <xsd:attribute name="resourceID" type="xsd:string"/>
  </xsd:complexType>

  <xsd:complexType name="MetricList">
    <xsd:sequence>
      <xsd:element name="metric" type="SingleMetric" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="SingleMetric">
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:double"/>
  </xsd:complexType>

</xsd:schema>
D.2.5  app.event

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://net.kwfgrid/dr/appevent"
    xmlns:tns1="http://net.kwfgrid/dr/events"
    xmlns:tns2="http://net.kwfgrid/dr/pu"
    xmlns:tns3="http://net.kwfgrid/dr/cr">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            Application Event Data Schema
            Version 0.1
            Copyright 2004-2005, Hong-Linh Truong. All rights reserved.
            Written based on application event data schema developed in SCALEA-G
                    (http://dps.uibk.ac.at/projects/scaleag)
        </xsd:documentation>
    </xsd:annotation>
    <xsd:import namespace="http://kwfgrid.net/dr/events"
        schemaLocation="events.xsd" />
    <xsd:import namespace="http://kwfgrid.net/dr/pu"
        schemaLocation="pu.xsd" />
    <xsd:element name="monitoringData" type="MonitoringData"/>
    <xsd:complexType name="MonitoringData">
        <xsd:sequence>
            <xsd:element name="pu" type="tns2:ProcessingUnit" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="events" type="EventList" minOccurs="0" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="dataTypeID" type="xsd:NMTOKEN" fixed="app.event" />
        <xsd:attribute name="resourceID" type="xsd:string" />
    </xsd:complexType>
    <xsd:complexType name="EventList">
        <xsd:sequence>
            <xsd:element name="event" type="tns1:SingleEvent" minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
D.2.6 wfa.event

Currently, workflow execution events, data type wfa.event, is expressed by using the generic event representation (see Section D.2.7). When describing workflow execution events, the attribute dataTypeID is set to wfa.event and the attribute resourceID is set to the workflow instance id. Information about workflow execution status such as activity name, resource on which the activity executes, etc., is specified through tuples of attribute name and value.

D.2.7 Generic events

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:tns1="http://net.kwfgrid/dr/events"
            targetNamespace="http://net.kwfgrid/dr/genericevent"
            xmlns:tns="http://net.kwfgrid/dr/genericevent"
>
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
        </xsd:documentation>
    </xsd:annotation>

    <xsd:import
        namespace="http://kwfgrid.net/dr/events"
        schemaLocation="events.xsd" />

    <xsd:element name="monitoringData" type="MonitoringData"/>
    <xsd:complexType name="MonitoringData">
        <xsd:sequence>
            <xsd:element name="events" type="tns1:SingleEvent"
                          minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="dataTypeID" type="xsd:NMTOKEN"/>
        <xsd:attribute name="resourceID" type="xsd:string"/>
    </xsd:complexType>
</xsd:schema>
```

D.2.8 path.bandwidth.achievable.TCP

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://net.kwfgrid/dr/bandwidthtcp">

    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            path.bandwidth.capacity.TCP data schema for SCALEA-G
            Copyright 2003, Hong-Linh Truong. All rights reserved.
            The name path.bandwidth.achievable.TCP is recommended by DAMED (http://www-didc.lbl.gov/damed)
        </xsd:documentation>
    </xsd:annotation>
</xsd:schema>
```
<xsd:element name="monitoringData" type="MonitoringData" />

<xsd:complexType name="MonitoringData">
  <xsd:sequence>
    <xsd:element name="source" type="xsd:string"/>
    <xsd:element name="destination" type="xsd:string"/>
    <xsd:element name="eventtime" type="xsd:long"/>
    <xsd:element name="bandwidth" type="xsd:double"/>
    <xsd:element name="nStream" type="xsd:integer"/>
  </xsd:sequence>
  <xsd:attribute name="dataTypeID" type="xsd:NMTOKEN"
    fixed="path.bandwidth.achievable.TCP"/>
  <xsd:attribute name="resourceID" type="xsd:string"/>
</xsd:complexType>

</xsd:schema>
E PERFORMANCE DATA QUERY AND SUBSCRIPTION LANGUAGE

E.1 PDQS SCHEMA

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://net.kwfgrid/dr/pdqs" >
    ...
</xsd:schema>
E.2 PDQS Data Aggregation

Currently PDQS allows to specify simple mechanism for data aggregation. Only two operators, MAX and MIN, are supported at this time.
Bibliography


    http://java.sun.com/j2se/1.5.0/docs/guide/jvmti.


