DIPAS: A Distributed Performance Analysis Service for Grid Service-based Workflows

Hong-Linh Truong [a,*]
Peter Brunner, Vlad Nae, Thomas Fahringer [b]

[a] *Distributed Systems Group, Vienna University of Technology*
Argentinierstrasse 8/184-1, A-1040 Vienna, Austria
truong@infosys.tuwien.ac.at

[b] *Distributed and Parallel Systems Group, Institute of Computer Science, University of Innsbruck*
Technikerstrasse 21A, A-6020 Innsbruck, Austria
{brunner,vlad,tf}@dps.uibk.ac.at

Abstract

Grid workflows are executed on diverse resources whose interactions are highly complex and hardly predicted. Often the user and the workflow middleware services want to be informed about the performance behavior of workflows, as early as possible, so that they can steer the execution of workflows to compensate the performance loss or execution failures. This paper describes a distributed performance analysis service that supports tracing execution, analyzing performance overheads, and searching for performance problems of Web services-based workflows in the Grid. We present how the user and the Grid workflow middleware can utilize the distributed performance analysis service in order to optimize the execution of workflows.

Key words: Grid computing, performance monitoring and analysis, performance problems, Grid workflows, Grid/Web services

1 Introduction

Unlike business workflows which are normally executed in stable environments, Grid workflows are prone to failures and performance variability because the Grid resources on which workflows are executed are dynamic and diverse; the

* Corresponding author. Email: truong@infosys.tuwien.ac.at

Grid is an unreliable and open environment. Furthermore, Grid workflows, currently most used for scientific computing, as shown in [1], are more ad-hoc, thus they imply a high degree of complex interactions. As a result, performance tools should detect performance problems and failures of Grid workflows during runtime and inform relevant parties, such as the user (the end-user or the developer) or the Grid workflow middleware (e.g., the workflow scheduler and the workflow execution service), of the problems and failures as early as possible. Moreover, different from conventional (non Grid-based) performance analysis tools which are targeted to expert users and developers who can detect performance problems and optimize their code, the performance analysis tools for Grid workflows must additionally support the Grid middleware services. It is because the Grid offers a huge amount of heterogeneous computing resources, making the traditional code optimization cycle difficult. Instead, the performance optimization is shifted to Grid middleware services which aim at optimizing the execution of Grid workflows by means of the optimization of resource selections and execution control, and self-adaptive techniques [2–8]. To support this paradigm shift, Grid middleware require performance analysis services which are capable of detecting performance problems of Grid workflows during runtime.

To fill the gap between the demand for workflow performance analysis services and the limited number of such services available, we have developed distributed performance monitoring and analysis services which aim at supporting online performance monitoring and evaluation of Grid workflows within the K-WfGrid project [9]. One of the K-WfGrid core components is a distributed performance analysis service (DIPAS) whose objectives are to help the user to understand the performance behavior of Grid workflows and to provide performance results to Grid workflow middleware for constructing and executing Grid workflows. In this paper, we describe DIPAS, its features, and how it assists the end-user, the developer, and the Grid workflow middleware to understand the performance of Grid workflows and to utilize the performance information for optimizing the execution of workflows. We also illustrate DIPAS through the performance evaluation of real-world workflows. This paper substantially refines and extends our previous conference paper published in [10], focusing on the workflow performance evaluation and providing an up-to-date status of the development of DIPAS.

The rest of this paper is organized as follows: Section 2 discusses the requirements for a performance analysis service for Grid workflows. Section 3 describes the architecture of the distributed performance analysis service. Section 4 discusses performance service interfaces, data representations, and request languages. Section 5 presents online performance analysis features. Experiments demonstrating DIPAS are given in Section 6. We outline the related work in Section 7. Section 8 summarizes the paper and discusses our future work.

2 Performance Analysis Requirements for Grid Workflows

2.1 K-WfGrid Objectives and User Classes

The main objective of the K-WfGrid project is to support semi-automatic, knowledge-based composition and execution of Grid workflows [9,11]. In K-WfGrid, Grid workflows are composed from Grid/Web services. The composition and execution of workflows relies on many types of knowledge, including domain-specific knowledge, application structure, available software resources and their relationship, and computing resources. The knowledge is assimilated from different services and managed by a distributed semantical knowledge base.

Two kinds of users must be supported in K-WfGrid. First, the application developer will exploit the K-WfGrid system to design, customize, develop and test workflows. Second, the application user will execute workflows provided by the developer. Both types of users require the performance of the Grid infrastructure and the Grid workflows being monitored and evaluated. While the application user requires online monitoring of the workflow execution and online notification of performance problems, the application developer requires a full support of performance evaluation, including post-execution analysis features, in order to efficiently construct and execute workflows.

Apart from these types of users, Grid middleware services, such as the workflow execution service and the workflow scheduler, also require performance information about executed and running activities to dynamically control the execution of workflows. In particular, information about Grid services and workflow performance will be used to estimate the performance of future Grid service invocations. The performance estimation is an important input for the composition and execution of future workflows.

2.2 Requirements for Performance Analysis in K-WfGrid

The above-mentioned objectives and user classes imply various requirements for the performance analysis support in K-WfGrid:

Supporting online and offline performance analysis. Runtime and offline performance information of workflows is not only an important source of knowledge but also provides necessary information for acquiring new knowledge about the performance of future Grid workflows. Grid workflows take a long time to finish and the resources are prone to failures. Therefore, both the user and the Grid workflow middleware require runtime performance evaluation

because they want to react as soon as possible to performance problems. For example, when the execution of a service invocation - within an activity instance of a long running workflow - takes a long time to finish due to the slow performance of the machine where the service is hosted, the workflow can be suspended and rescheduled in order to overcome the performance problem. Moreover, the application developer needs to examine in detail the performance of workflows as well as to capture performance information for further use. This requires the user to spend a certain amount of time to carefully examine all performance behavior. Such a required time may not be enough during runtime of a workflow. Thus, the user should also be able to conduct the performance analysis of completed workflows. This feature is useful for developers as they can run several experiments of their workflows and select specific experiments for performance study. Furthermore, it allows for the assimilation of performance knowledge for performance estimations.

Allowing specifying performance requests before and during the execution. Since the client of the performance analysis service needs to react to performance problems immediately, the client of the performance analysis services must be able to prepare performance analysis tasks, before and during runtime. For example, at a given time during the workflow execution, the user can select an activity which has not been started yet and specify analysis tasks for that activity. Based on that, the performance analysis service can prepare necessary steps. When the activity is instantiated, performance analysis techniques can be applied to the activity instance immediately.

Providing highly interoperable interfaces for Grid workflow middleware to specify analysis requests and to obtain performance result. Performance analysis services have to be loosely integrated with various services and different performance requests and information are exchanged between clients, middleware and performance services in the Grid environment. To make sure that Grid workflow middleware and clients can seamlessly be integrated and can interoperate with the performance analysis services, the analysis services must offer highly interoperable interfaces, e.g., based on WSRF (Web Services Resource Framework [12]), for other services and clients to utilize their performance analysis features. Furthermore, all performance data as well as analysis requests should also be well defined, using open, standardized representations, such as XML.

Our DIPAS system supports all the above-mentioned requirements. DIPAS offers WSRF interfaces, providing XML schemas for performance data and analysis requests. As a result, any clients and services can easily request performance results from DIPAS, based on WSRF and XML. Moreover, DIPAS interacts with other components through a loosely coupled WSRF-based interface, thus it can readily be integrated with other Grid workflow middleware services.

3 The Distributed Performance Analysis Service

3.1 Overview

Figure 1 depicts the architecture of our distributed performance analysis service (DIPAS) and relevant components involved in the performance analysis. DIPAS basically includes two parts: (i) DIPAS Portal, a portal for the end-user and developer, and DIPAS Client, any clients that need performance results, and (ii) DIPAS Gateway, the core service performing online overhead analysis and search for performance problems. DIPAS interacts with underlying workflow execution, monitoring and knowledge core services that control and monitor the execution of workflows.

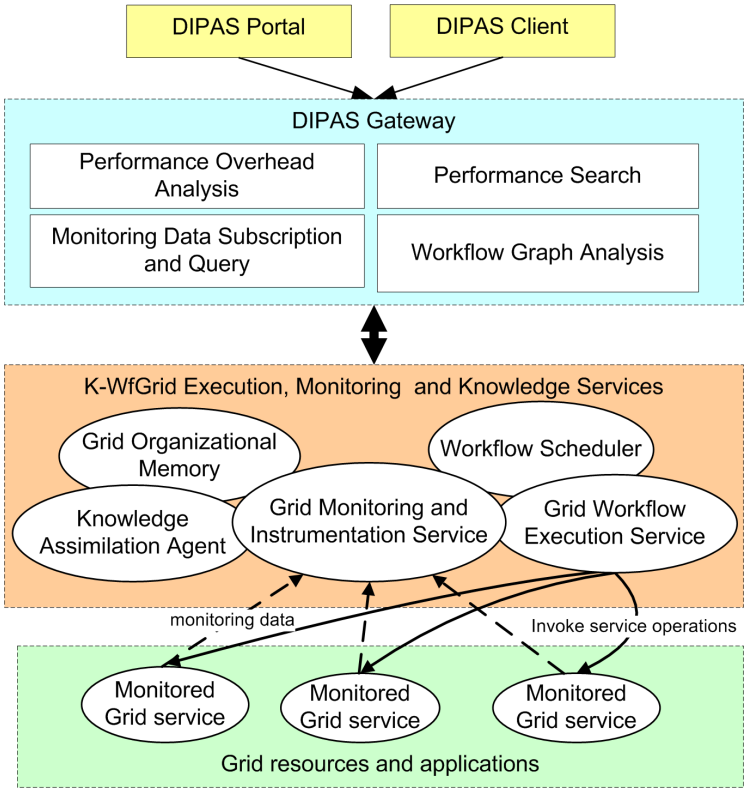

Fig. 1. Overall architecture of the distributed performance analysis service (DIPAS). Arrowed black lines indicate the invocation of service operations (control and data flow). Arrowed dash lines indicate data flows.

DIPAS Gateway supports the performance evaluation by analyzing monitoring data, workflow representation, and client's analysis requests. The monitoring data is given by the Grid Monitoring and Instrumentation Service (GEMINI) [13] whereas the workflow representation is given by the Grid Workflow Execution Service (GWES) [14]. DIPAS Gateway discovers workflow events by accessing information describing events in the Grid Organizational Memory

(GOM) [15]. DIPAS Gateway includes a component to query and subscribe online monitoring data - *Monitoring Data Query and Subscription* - and a component to analyze workflow representations - *Workflow Graph Analysis*. Based on these two components, a component named *Performance Overhead Analysis* will conduct the performance evaluation of workflows. This component determines performance overheads based on a well-defined classification of workflow performance overheads [16]. The *Performance Search* component will perform the search and check performance problems based on performance conditions specified by clients. DIPAS also provides information for other Grid middleware services, such as the Workflow Scheduler - which uses the information to schedule the execution of activities - and the Knowledge Assimilation Agent (KAA) - which utilizes runtime and past performance information to estimate the performance of future Grid services and workflows.

To make sure that DIPAS supports both the user (the application developer and user) and the Grid middleware, DIPAS Gateway provides WSRF operations for any clients to conduct the analysis. More importantly, the requests and responses of any performance analysis tasks are defined using a set of XML-based languages. Currently, DIPAS implements and supports WARL (Workflow Analysis Request Language) and the result of performance analysis is returned in a pre-defined XML representation. DIPAS provides two tools - an applet in a Web portal and a Java application - to the user to conduct the performance analysis. Other Grid workflow middleware interact with DIPAS using WSRF-based operations and XML-based data representations. DIPAS Gateways publish their information into a registry (currently, we use GOM and OpenDHT [17], an open service supporting distributed hash tables). Thus, in addition to manual configuration, any client can search available DIPAS Gateways from the registry. Currently, monitoring data of a single workflow can be provided by multiple GEMINI services, but the performance evaluation of a single workflow is entirely carried out at a single DIPAS Gateway which subscribes or queries monitoring data from multiple GEMINI services. The deployment of multiple DIPAS Gateways is useful in multi-user environments.

DIPAS is a part of the K-WfGrid workflow system [11] and has been integrated with GEMINI [13], GWES [14], GOM [18], the Workflow Scheduler, and KAA. All of them are Grid/Web services, offering WSDL or WSRF interfaces. Typical steps in the execution and performance evaluation of a workflow in the K-Wfgrid workflow system are given as follows:

- The workflow description is submitted to GWES and GWES returns a workflow ID to the submitter (a user or a middleware service). GWES starts to enact the workflow. GWES executes workflow activities and, in turn, service operations of Grid/Web services are invoked.
- GEMINI monitors both GWES and invoked Grid/Web services. Monitoring data of the workflow and invoked service operations are captured in events

and managed in GEMINI. Information about available types of events and GEMINI services are sent to GOM.

- Using the workflow ID, the user can select a performance evaluation feature from a DIPAS Portal which will generate a performance request or a middleware service can issue a performance request to a DIPAS Gateway. If it has not been done before, the DIPAS Gateway will use the workflow ID to query GOM to find GEMINI services which provide monitoring data, subscribe/query the monitoring data from appropriate GEMINI services, and obtain the workflow description from GWES. Based on the request, the subscribed/queried monitoring data, and the workflow description, the performance evaluation is performed, and the results are returned to or polled by the user or the middleware service.

3.2 Supported Grid Workflows

DIPAS supports the performance analysis of K-WfGrid workflows which are described by GWorkflowDL [19]. GWorkflowDL is a Petri net based representation in which transitions are used to describe workflow activities. GWorkflowDL is capable of modeling multiple levels of abstraction of workflows, e.g., a transition in GWorkflowDL can be used to represent a user request which later on is mapped to an abstract operation. Each abstract operation can be associated with multiple service operation candidates, of which one concrete service operation will be selected and executed by the execution service. The concrete service operation is a Web services operation or a WSRF operation.

Workflows in K-WfGrid are executed by GWES [14] which is able to execute workflows of Web services and of WSRF ones. GWES is instrumented and monitored by GEMINI. Through the monitoring, all execution statuses of workflows and workflow activities will be captured and represented in events which can be subscribed and queried during runtime of workflows. DIPAS provides analysis and visualization capabilities for workflows at runtime by processing workflow/activity events, workflow representations, and performance constraints specified by its clients.

3.3 Interactions between Clients and DIPAS

Figure 2 shows a simplified sequence diagram illustrating the interactions between clients and DIPAS. *WfPerfComponent* dispatches requests coming from different clients to corresponding *PerfOverheadAnalysis* instance, for performance overhead analysis, and to *PerfProblemSearch* instance, for search for performance problems. Note that whether a request will be processed by *PerfOverheadAnalysis* or by *PerfProblemSearch* is dependent on the content of

the request, although all requests are expressed by the WARL. At a given time, performance overheads and problems of multiple workflows can be evaluated: an instance of *WfPerfComponent* is responsible for a workflow. Each instance will dynamically compute overhead metrics and determine if there exist performance problems by examining performance conditions specified in the requests. The performance metrics and problems are determined during runtime of workflows but the result is buffered at the DIPAS Gateway. Then, the client can poll the DIPAS Gateway to receive the results. Performance problems are described in XML by using the `appprof` schema which can describe, in detail, all performance metrics associated with workflows.

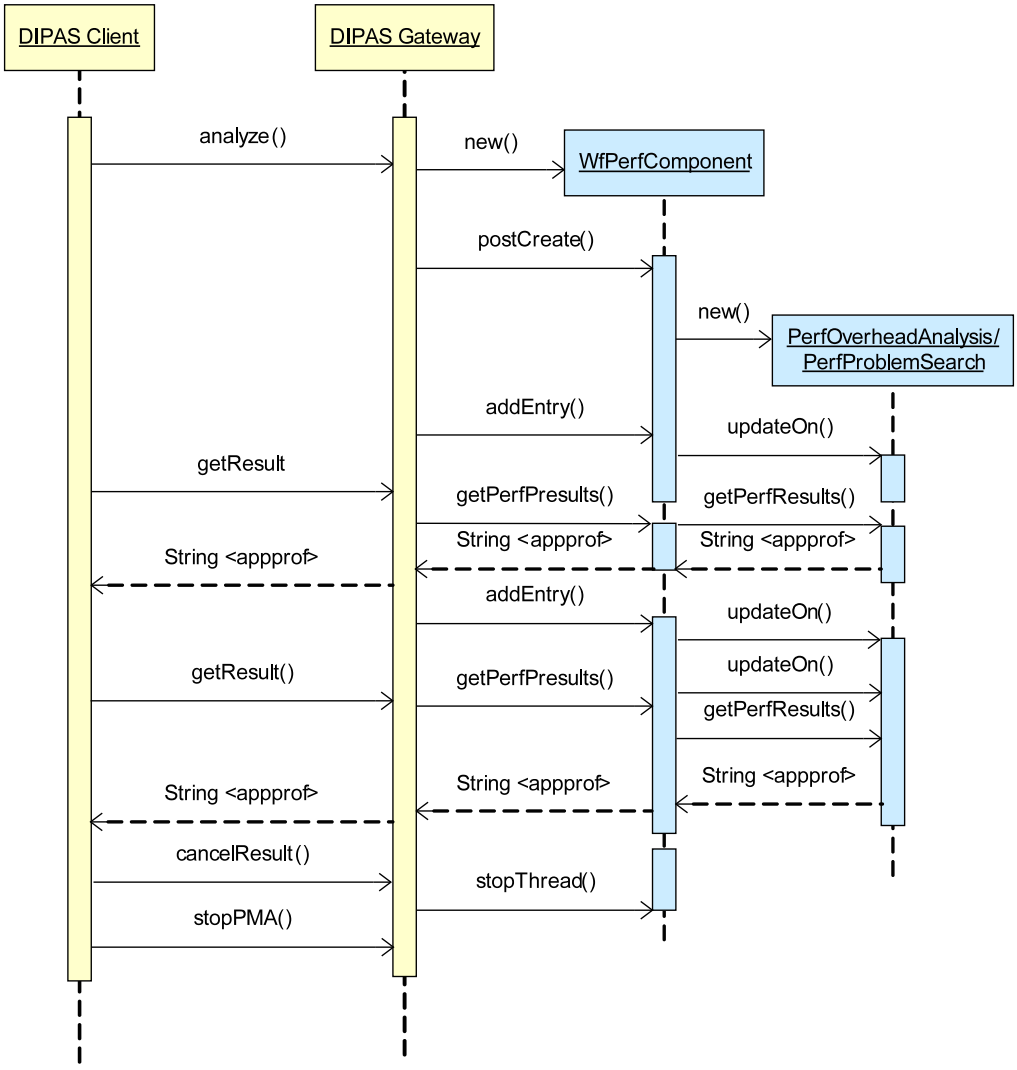

Fig. 2. A simplified UML sequence diagram of interactions between a DIPAS client and a DIPAS Gateway. Arrowed black lines indicate invocations of service operations or functions. An arrowed dash line denotes returning data of an invocation and data format.

The above-mentioned model supports multiple clients analyzing the perfor-

mance of multiple workflows. Although, currently, the performance analysis for a single workflow is conducted at a single place, distributed DIPAS Gateways can be used and the workflow events are collected from various places. Therefore, DIPAS is capable of supporting distributed workflows.

4 Performance Service Interfaces, Data Representations and Request Languages

4.1 Retrieving Monitoring Data from Monitoring Service

To analyze the performance of a workflow, DIPAS relies on the representation of the workflow and events collected and propagated to DIPAS on the fly. To ensure that DIPAS can seamlessly process and utilize various monitoring data types of different monitored resources as well as can be easily adopted for other workflow systems, we represent all monitoring data in XML format.

In K-WfGrid, DIPAS retrieves online monitoring data from GEMINI which collects monitoring data from various sources. In order for a client to locate monitoring services which provide monitoring data, GEMINI publishes information about available monitoring data into GOM, an ontology-based knowledge repository. Information about monitoring data is described in OWL (Web Ontology Language) and is published into GOM. From GOM, DIPAS knows which GEMINI services it should contact in order to obtain the monitoring data of a particular workflow. In addition, using the information provided by GOM, DIPAS can build requests which are sent to GEMINI to retrieve the monitoring data.

Based on our extensible event schemas, we have developed a generic performance data query and subscription (PDQS) language for querying and subscribing various types of monitoring data. Figure 3 shows the schema of PDQS. Basically, we classified monitoring data into differen types, such as workflow monitoring data or activity monitoring data. Each type of monitoring data and each monitored resource (e.g., an activity or a workflow) are associated with a unique *dataTypeID* and *resourceID*, respectively. Based on that, the client can specify PDQS requests which include *(dataTypeID,resourceID)* together with other information like XPath/XQuery-based data filters (denoted by *dataFilter*) and subscription time (denoted by *subscriptionTime*). A PDQS request can be used to *query* or *subscribe* a particular type of monitoring data of various resources.

PDQS requests can be built based on the published information in GOM. For example, Figure 4(a) presents an OWL description for workflow events of the

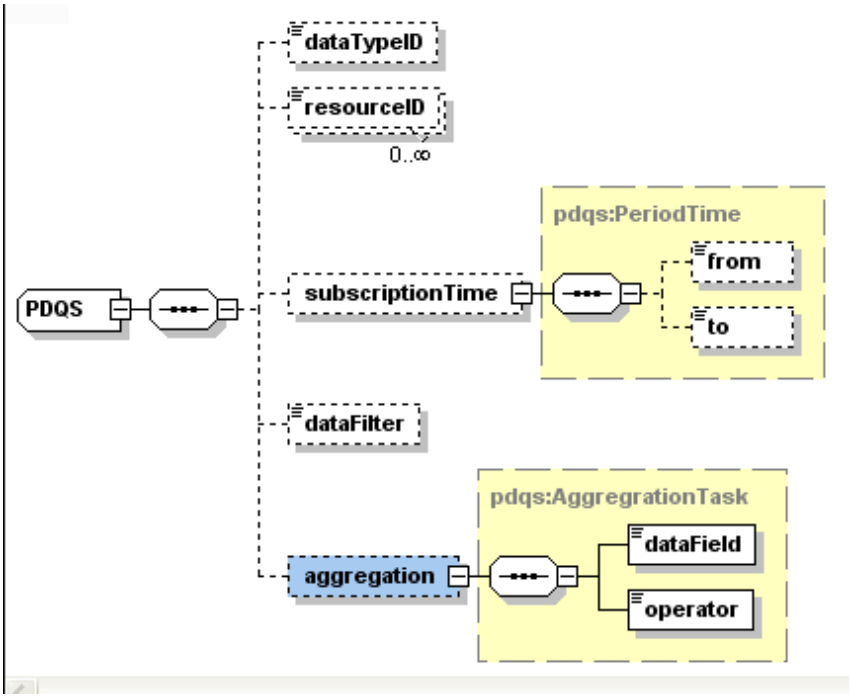

Fig. 3. The schema of the performance data query and subscription (PDQS) language visualized with the `XMLSpy` tool

workflow whose workflow ID is `truong_810cf130-eb24-11da-8ebd-a46bfd55290e`. The GEMINI services that provide these events can be obtained by using the information indicated by the tag `<dg:isStoredIn>`. Figure 4(b) presents the corresponding PDQS request used to subscribe all workflow events generated during the workflow's execution by specifying the workflow ID and setting the subscription time to 0 (until the end of the execution).

4.2 Workflow Analysis Request Language

One of the goals of DIPAS Gateways in Figure 1 is to support user-defined search for performance problems and to inform clients about the detected problems. To this end, we have utilized performance metrics associated with multiple levels of Grid workflows defined in [20]. Performance metrics include, for example, overheads (e.g., middleware overhead, due to scheduling or resource management, or loss of parallelism overheads, due to load imbalance), execution time, and data movement. The performance of K-WGrid workflows is systematically analyzed according to these metrics. By using pre-defined metric names, performance problems can then be determined based on conditions established on the basis of appropriate performance metrics (for example, overheads, execution time, and start/end time), and their thresholds. A performance problem occurs when a condition is met. During the execution of the workflow, any clients of DIPAS can specify requests to obtain performance

```
<dg:DataObject rdf:ID="MD1148476305524_DO">
 <dg:contains>
    <dg:MonitoringData rdf:ID="MD1148476305524">
      <dg:hasDataType rdf:datatype="...">wfa.event
      </dg:hasDataType>
      <dg:ofResource rdf:datatype="...">
       truong_810cf130-eb24-11da-8ebd-a46bfd55290e
      </dg:ofResource>
      <dg:validFrom rdf:datatype="...">1148476305524
      </dg:validFrom>
      <dg:validTo rdf:datatype="...">0</dg:validTo>
    </dg:MonitoringData>
 </dg:contains>
 <dg:isStoredIn rdf:resource="http://gom.kwfgrid.net/gom/
  ontology/ServiceRegistry/CMN#MSa6240bba-3c48-4cc6-ad31
  -648e9b60124b"/>
</dg:DataObject>
```

(a) an OWL-based description (simplified) of workflow events

```
<pdqs xmlns="http://net.kwfgrid/dr/pdqs">
   <dataTypeID>wfa.event</dataTypeID>
   <resourceID>truong_810cf130-eb24-11da-8ebd-a46bfd55290e
   </resourceID>
   <subscriptionTime>
      <from>0</from>
      <to>0</to>
   </subscriptionTime>
</pdqs>
```

(b) the corresponding PDQS request

Fig. 4. An example of OWL description and PDQS request for workflow events

metrics and to check performance problems.

To simplify the interaction between clients and DIPAS Gateways and to support various types of clients such as the end-user, the developer and the workflow middleware, we design a novel workflow analysis request language (WARL) which is used to specify analysis requests. Figure 5 presents the current version of WARL. A WARL request includes three parts: constraints (element `constraint`, type `WARLConstraint`), performance metrics to be analyzed (element `analyze`, type `WARLAnalyze`), and performance conditions (element `perfProblemSpecs`, type `WARLPerfProblemSpecs`). Constraints include information about hierarchical workflow concepts [20] (e.g., `Workflow`, `WorkflowRegion` and `Activity`) to be analyzed and their properties (e.g., ID of `WorkflowRegion` and the name of `Activity`). Each concept is identified by a name and a type: the name indicates the identifier of the concept in the workflow description, e.g., activity name, while the type deter-

mines whether the concept is, e.g., a workflow, an activity, or a workflow region. A WARL `analyze` request specifies a list of performance overheads that should be analyzed and provided. Performance problems that should be monitored and detected are specified by using WARL `perfProblemSpecs`. Performance problems can be checked by specifying a set of performance conditions, each condition includes a metric name, an operator (e.g., greater than or less than), and a value (e.g., indicating a threshold). Given WARL requests, the performance analysis service will conduct corresponding analyses and send to the requesters the analysis result described in XML. Figure 6 presents an example of a WARL request. This request indicates that a client needs to analyze performance metrics named `LoadIm, TotalOverhead` and `QueuingTime`, and to be informed about performance problems related to `SynDelay` and `QueuingTime` of three activities `computeStartZonePolyg`, `computeEndZonePolyg, computeStartNodes` and the workflow `truong_3d6c4330-eb2a-11da-8ebd-a46bfd55290e`. By using WARL to represent such a request, performance requests can be defined by both the user and the middleware services. WARL can also facilitate the integration among performance clients and different performance analysis services by acting as a common performance request representation.

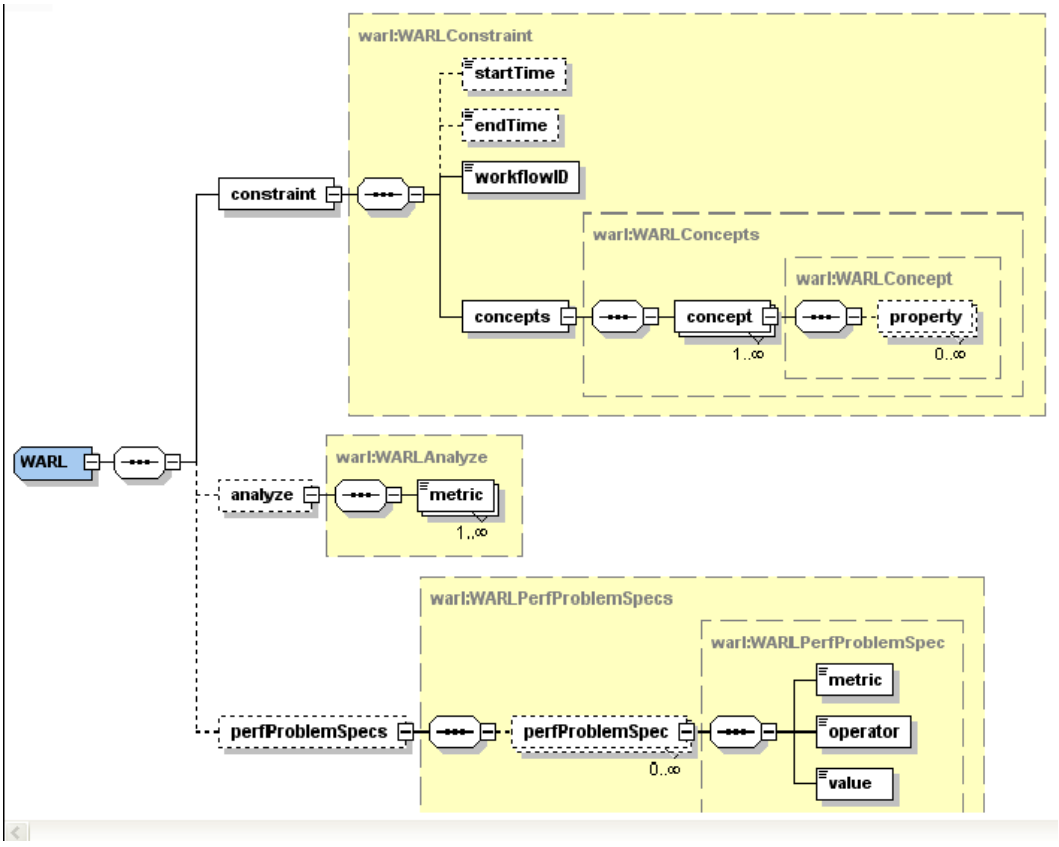

Fig. 5. Workflow analysis request language (WARL) visualized with the `XMLSpy` tool

```
<WARL>
<constraint>
  <startTime>0</startTime>
  <endTime>0</endTime>
  <workflowID>truong_3d6c4330-eb2a-11da-8ebd-a46bfd55290e
  </workflowID>
  <concepts>
   <concept
       name="truong_3d6c4330-eb2a-11da-8ebd-a46bfd55290e"
        type="Workflow"/>
   <concept name="computeStartZonePolyg" type="Activity"/>
   <concept name="computeEndZonePolyg" type="Activity"/>
   <concept name="computeStartNodes" type="Activity"/>
  </concepts>
</constraint>
<analyze>
   <metric>LoadIm</metric>
   <metric>TotalOverhead</metric>
   <metric>QueuingTime</metric>
</analyze>
 <perfProblemSpecs>
   <perfProblemSpec>
        <metric>SynDelay</metric>
        <operator>GE</operator><value>20</value>
   </perfProblemSpec>
   <perfProblemSpec>
        <metric>QueuingTime</metric>
        <operator>GE</operator><value>10</value>
   </perfProblemSpec>
 </perfProblemSpecs>
</WARL>
```

Fig. 6. Example of a WARL request for obtaining performance overheads and specifying performance conditions

5 Online Performance Analysis Features

The online performance analysis is centered on (i) elements of workflows to be analyzed, e.g., *workflow, workflow region, activity* and *activity instance*, and (ii) performance metrics and statuses, e.g., *elapsed time, queuing time* and *start time*, associated with these elements. These elements and metrics are defined in the performance ontology for Grid workflows [20].

From the workflow representation, the *Workflow Graph Analysis* component will produce a hierarchical structure view of workflows [20]: the workflow will consist of workflow regions, each region has a set of activities. Basically, a

workflow region reflects a workflow pattern [21] that includes set of activities structured in a single-entry-single-exit subgraph. Supporting analysis of workflow regions is an important feature as the client is normally interested in workflow regions which refer to workflow patterns used to implement particular solutions. In K-WfGrid, Petri net-based workflow representation is used, and this representation does not explicitly include constructs describing workflow regions. Our *Workflow Graph Analysis* parses a Petri net representation and produces the structured view of the workflow.

Each activity can have different instances. DIPAS supports the performance evaluation of the workflow, the workflow region, the activity, and the activity instance. In the following sections, we present the main analysis features.

5.1 Execution Tracing

Tracing application execution is a popular method, e.g., in parallel computing, used for studying complex interactions within applications. However, execution tracing is not well supported in contemporary Grid workflow performance tools. Within DIPAS, workflow trace events are analyzed and visualized, allowing us to examine, in detail, *execution phases* of every activity instances and where the activity instances are executed. Workflow trace events contain various types of data related to activities, activity instances, and their runtime behavior. For example, events consist of *activity name*, *activity type* (e.g., computational or data transfer tasks), *activity instance ID*, *Web services operation name*, *URL of the WSDL of the Web services*, *computational node*, etc. With workflow tracing, the user can examine states of activity instances and workflows (e.g., initialized, active or terminated), execution phases of activity instances and workflows (e.g., in processing, queuing, or suspending), and performance status of computational nodes used.

Based on tracing information, execution of activities and workflow regions can be studied. Moreover, various analyses such as load imbalance, activity distribution, etc., are also supported. In our framework, tracing execution of workflows can also be applied to historical workflows, provided that the monitoring middleware still keeps the workflow events. The tracing feature is intended only for the application user and developer to analyze their workflows. This feature is largely based on visualizations, thus it is not suitable for the middleware. However, the Grid middleware can retrieve the trace-based monitoring data by subscribing the monitoring service.

The performance overhead analysis is conducted based on a systematic classification of workflow overheads. The workflow overheads [16] can be due to, for example, application structures (e.g., parallel synchronization) and middleware (e.g., scheduling). DIPAS supports workflow overheads at multiple levels of abstraction, ranging from the workflow to the workflow region to the activity. The overhead analysis is targeted to both the user and middleware.

```
<WARL>
  <constraint>
    <startTime>0</startTime>
    <endTime>0</endTime>
    <workflowID>truong_153bd250-73d2-11db-8062-ef29e6a62bc8
    </workflowID>
    <concepts>
       <concept name="computeSSSP" type="Activity"/>
       <concept name="Sequence0" type="WorkflowRegion"/>
       <concept name="Sequence2" type="WorkflowRegion"/>
       <concept name="Loop0" type="WorkflowRegion"/>
    </concepts>
  </constraint>
  <analyze>
     <metric>ALLOVERHEAD</metric>
  </analyze>
</WARL>
```

Fig. 7. A WARL request for obtaining workflow overheads.

While the user can use DIPAS portal to analyze the workflow overhead, the Grid workflow middleware can easily specify analysis requests and receive performance overheads by using a few lines of code. Figure 7 illustrates a simple WARL request used to analyze performance overheads of a workflow and Figure 8 presents a code excerpt that a client can use to obtain the overheads. The analysis request in Figure 7 specifies the workflow to which the analysis should be applied (by using `workflowID` element), how long the request should be conducted (by using `startTime/endTime` element), which workflow activities and regions should be analyzed (by using `concept` element) and which overhead metrics should be provided using `metric` element). In this example, we requested the analysis of all overhead metrics (denoted by a metric named `ALLOVERHEAD`) for three workflow regions named `Sequence0, Sequence2` and `Loop0`, and one activity named `computeSSSP`. The list of overhead metrics is well-defined in DIPAS and can be specified by the client, together with the list of selected workflows, workflow regions and activities. The analysis time associated with a request is required as the overheads of workflows are changed during the execution of the workflows and the client may need to be informed

of that change over the time. In this example, as `startTime/endTime=0` the request will be analyzed until the workflow finishes or until the client cancels the request. The client can obtain overhead results, described in an XML representation, by polling the DIPAS Gateway. Figure 8 shows a code excerpt for sending the requests and polling the results.

By supporting WSRF-based operations and XML-based data representations and requests, DIPAS facilitates the integration between Grid middleware services and performance analysis services.

```
//select a DIPAS Gateway through an endpoint reference
DIPASServiceAddressingLocator
    locator = new DIPASServiceAddressingLocator();
final DIPASPortType
    resource = locator.getDIPASPortTypePort(dipasEndpoint);
...
//create a new analysis request
AnalysisRequestMessage
    message = new AnalysisRequestMessage();
//indicate this is a WARL request, instead of PDQS request
message.setLanguage("WARL");
//specifying the content of request which include
//performance conditions. warlRequest is an XML string based
//on WARL schema
message.setContent(warlRequest);
...
//invoke DIPAS with the input request and get back the
//resultID which is used to poll the result
final String resultID = resource.analyze(message);
...
//using resultID to get results of a request
//result will be returned in XML.
DataEntries entries = resource.getResult(resultID);
```

Fig. 8. Code excerpt of using DIPAS to obtain workflow overheads

5.3 Search for Performance Problems

The search for performance problems is intended for both the user and the Grid middleware. The search for performance problems is conducted at DIPAS Gateways based on performance conditions, online performance metrics and events, and workflow graphs. Performance search allows the client to specify performance conditions associated with workflows, workflow regions and activities.

Performance conditions are defined based on performance metrics (e.g., elapsed time, processing time, and queuing time) and other inputs (e.g., the latest start time and the machine name). The WARL is also used for specifying performance conditions. Clients that want to be informed of performance problems can use similar steps for obtaining overhead metrics (in Section 5.2) to specify performance conditions and to receive performance problems, if occurred. In this case, the WARL will include a set of performance conditions, instead of overhead metrics, associated with the workflow, and workflow regions and activities. An example of performance conditions is given in the following:

```
<WARL>
<perfProblemSpecs>
    <perfProblemSpec>
        <metric>ElapsedTime</metric>
        <operator>GE</operator>
        <value>30</value>
    </perfProblemSpec>
</perfProblemSpecs>
</WARL>
```

A performance condition, specified by a `perfProblemSPec` tag, indicates that there is a problem when the elapsed time (denoted by `ElapsedTime`) is greater than `30` seconds. Performance problems reported will be described in XML. The middleware services can specify WARL requests and use a similar code to the excerpt in Figure 8 to obtain existing performance problems.

6 Experiments

We have implemented DIPAS entirely in Java using GT 4.0 [22], Gridsphere [23], JFreeChart [24] and JGraph [25]. Figure 9 depicts our performance monitoring and analysis portal for Grid workflows. From the portal the user can select existing workflows, currently being executed by or submitted to GWES, and start the monitoring and analysis. The workflow representation is shown in the left-pane whereas online execution progress of the workflow and its activities, together with performance metrics, are shown in the right panes. During runtime, the user can conduct performance evaluation and search by selecting activities or activity instances and features in the menu.

We use the most up-to-date version of the Coordinated Traffic Management (CTM) workflow developed within the K-WfGrid project [26] for demonstrating analysis features that have been mentioned in previous sections [1]. The

[1] We also conducted the performance monitoring and analysis for two other

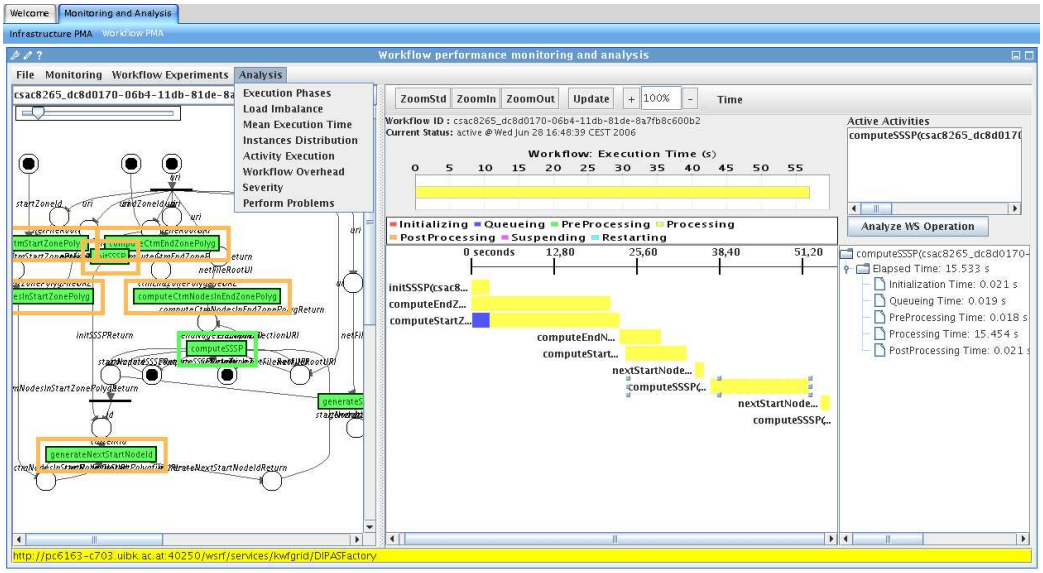

Fig. 9. Workflow performance monitoring and analysis portal. Execution statuses of activities and activity instances are visualized by different colors.

CTM workflow is used to calculate the best routes between two city districts, traffic flows among paths, and air pollutant emission. Figure 10 presents the graph of the CTM workflow. DIPAS analyzes the Petri net-based graph and produces workflow regions shown in Figure 10. By selecting a workflow region or activity in the `Workflow Regions` window, the corresponding subgraph of the selected region or activity will be shown in the `Workflow Visualization` window. The CTM workflow is composed from Grid services deployed in Bratislava, Cracow, Genova, and Innsbruck.

6.1 Execution Tracing

Figure 11 shows a snapshot of the tracing tool. All detailed execution phases, performance metrics as well as machine information are shown. By using this tool, we could spot some strange behavior. For example, in Figure 11, the first instances of activities `computeStartZonePolyg` and `computeEndZonePolyg` are initialized at the same time (because these activities can be executed in parallel, as shown in `Workflow Regions` window in Figure 10). However, `computeEndZonePolyg` was running only after `computeStartZonePolyg` finished. It is a strange behavior because they should be executed in parallel. We found that the Workflow Scheduler has serialized the two instances when map-

real-world workflows named *Flood Forecasting Simulation* and *Enterprise Resource Planning*. Recorded movies illustrating the performance monitoring and analysis of these workflows and the CTM workflow can be found at `http://www.dps.uibk.ac.at/projects/kwfgrid`

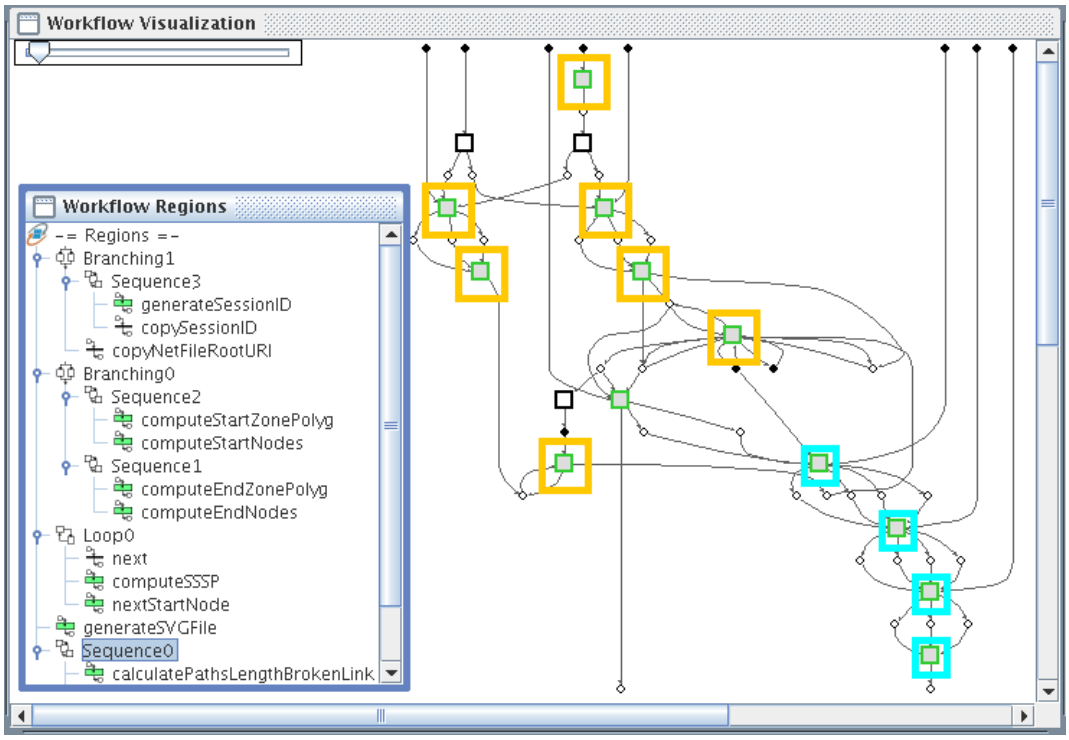

Fig. 10. Workflow regions of CTM. The Petri net graph of the workflow is visualized in the large pane. The window `Workflow Regions` shows the hierarchical structure of workflow regions. Specific regions are loop, branches and sequences. A workflow region may include sub regions. Only the leaf node in the tree denotes an activity.

ping them on the same machine. The same problem also occurred in the second instances of `computeStartZonePolyg` and `computeEndZonePolyg`. During runtime, using execution tracing tool, if the user detects some strange performance behavior, then the user can suspend the workflow, examine the execution status, and resume or terminate the execution of the workflow. Moreover, by using the detailed information about machines used in workflows, further infrastructure monitoring and analysis could be conducted through the integration with an infrastructure monitoring service [10].

6.2 Performance overhead analysis

Figure 12 shows an example of performance overhead analysis. The client can select any workflow activities and regions, and specify overhead metrics that should be determined. DIPAS will automatically compute the overheads during runtime. An overhead metric may be composed from other metrics. In this case, overheads are presented in a tree: a composed metric includes a breakdown of other metrics. In Figure 12, detailed execution time and overheads of all activity instances, regions and the whole workflow are presented. For example, it shows that one instance of `computeStartZonePolyg` has a large

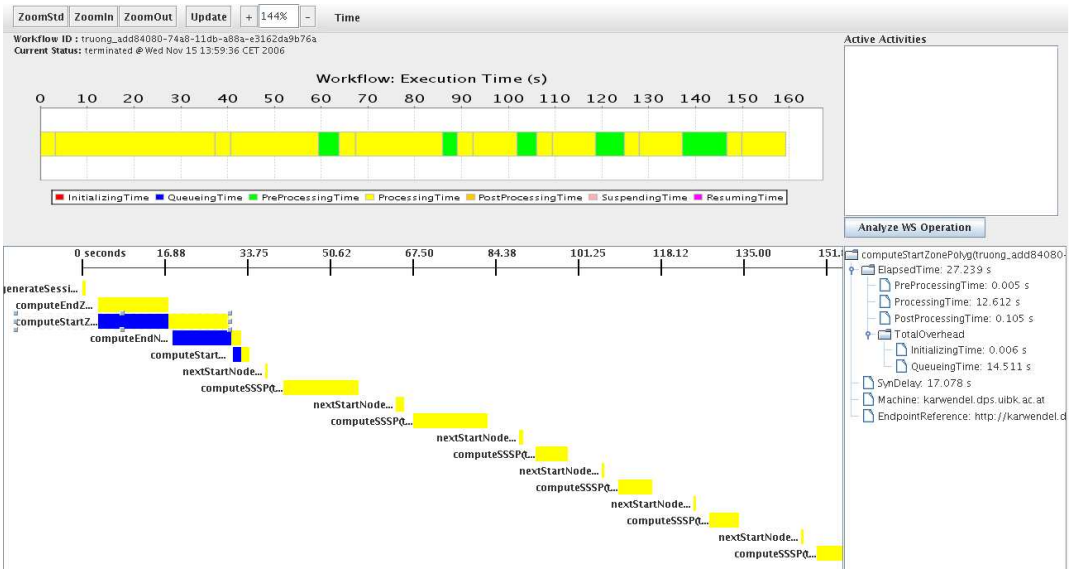

Fig. 11. Execution tracing of CTM workflow. The top-left pane visualizes the execution phases of the workflow. The bottom-left pane shows execution phases of activity instances. The bottom-right pane presents detailed performance metrics and machine information.

amount of elapsed time in which overheads hold more than 50% of the total time. Using the breakdown tree, we discovered that queuing problem is the main source of overheads. Then, for example, the scheduler can decide not to use the same machine in the next run of `computeStartZonePolyg`.

6.3 Search for Performance Problems

To experiment with the search for performance problems, we specified several performance conditions based on our expected performance of the CTM, given a problem size. Some conditions are listed below:

- is there any activity instance or region that takes more than 20% of total execution time (e.g., `ElapsedTime > 30` seconds)?
- is there any activity instance or region that spends more than 30% of its time in the queue (e.g., `QueuingTime > 10` seconds)?
- is there any activity or region that its average execution time per instances takes more than 10% of the execution time of the workflow (e.g., `MeanElapsedTime > 15` seconds).
- have all instances of activity `computeSSSP` been executed before a specified time (e.g., `startTime < Wed Nov 15 15:50:51 CET 2006`).

Such simple performance conditions are normally requested by the user and the Grid workflow middleware. For example, conditions on the start/end time of activities can be used to check whether advanced reservations on the ma-

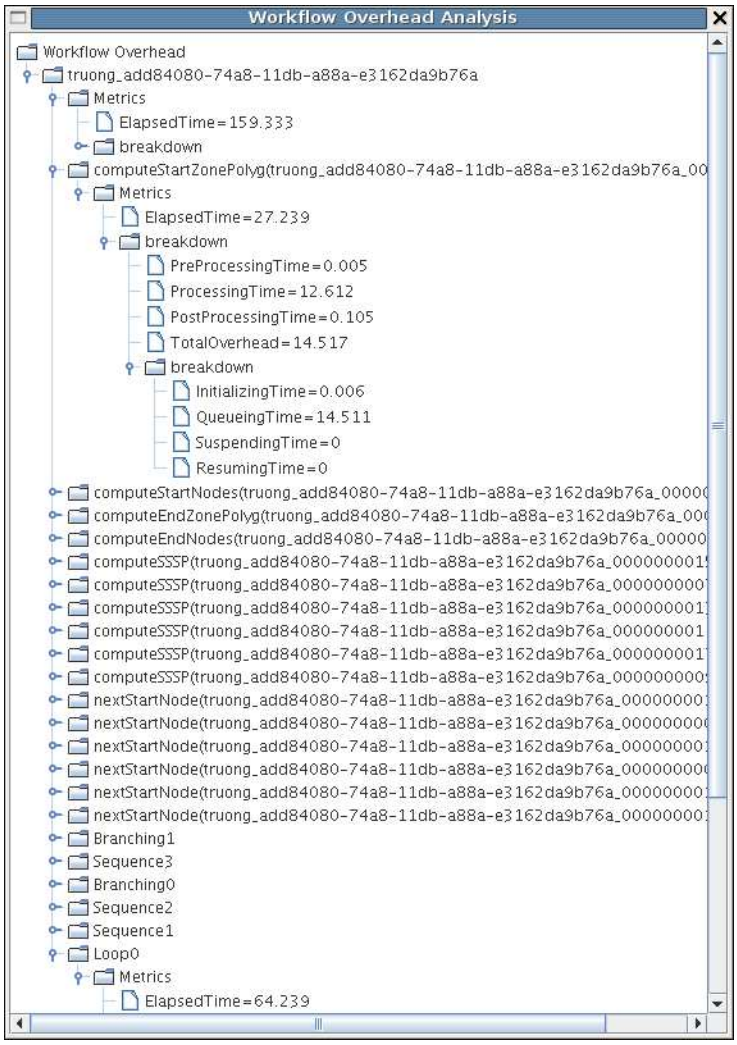

Fig. 12. Workflow overhead analysis for CTM. Nodes under the tree root represent instances of workflow, workflow regions and activities. Under a node representing an instance is a subtree visualizing performance metrics.

chine are fulfilled. In DIPAS, such conditions can be specified before the workflow is *running* (but after submitted), during runtime of the workflow, or even after the workflow is completed. Figure 13 presents how the user can specify performance conditions by using the portal. Figure 14 shows performance problems detected during one execution of the CTM workflow, given the above-mentioned performance conditions.

7 Related Work

The issues of tracing execution of applications, searching for performance problems, overhead analysis are previously addressed in parallel computing [27–29].

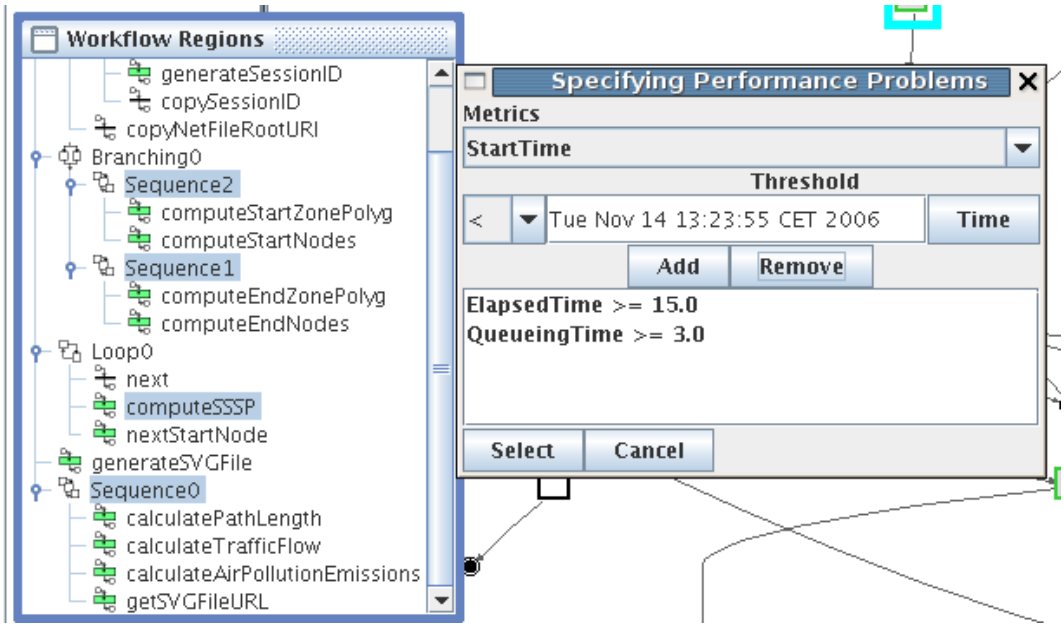

Fig. 13. Example of specifying performance problems. Workflow regions and activities are selected in the left window. In the right window, the user can define performance problems.

However, such similar features for Grid workflows are not well supported by existing Grid performance tools. Automatic search for performance problems are supported by various tools such as Paradyn [28] and AKSUM [29]. First of all, these tools focus on supporting the developer to find performance problems in order to tune their code. Secondly, these tools support parallel programs which are executed on dedicated environments. Because of differences in targeting clients and in operating environments, DIPAS differs significantly from these existing tools with respect of service interfaces and data representations. For instance, DIPAS is based on the service-oriented architecture model, and supports XML data representations and requests, making Grid workflow middleware to easily utilize DIPAS.

Many techniques have been introduced to study the quality of service and performance models and to monitor the execution of business Web services and workflows [2,30–32]. The WebLogic Process Integrator [33] allows the user to examine status of workflow instances. However, its monitoring is limited to the activity level. Web Service Navigator [34] provides visualization techniques for Web service based applications. ARM defines means for obtaining monitoring data of business transactions through instrumentation [35]. However, ARM agents are monitoring sensors that could be integrated into our framework. Performance search and analysis at multiple levels of abstraction are not supported in the business domain.

In the Grid domain, there is a lack of monitoring and analysis tools for workflows of Grid/Web services. P-GRADE [36] supports tracing of workflow appli-

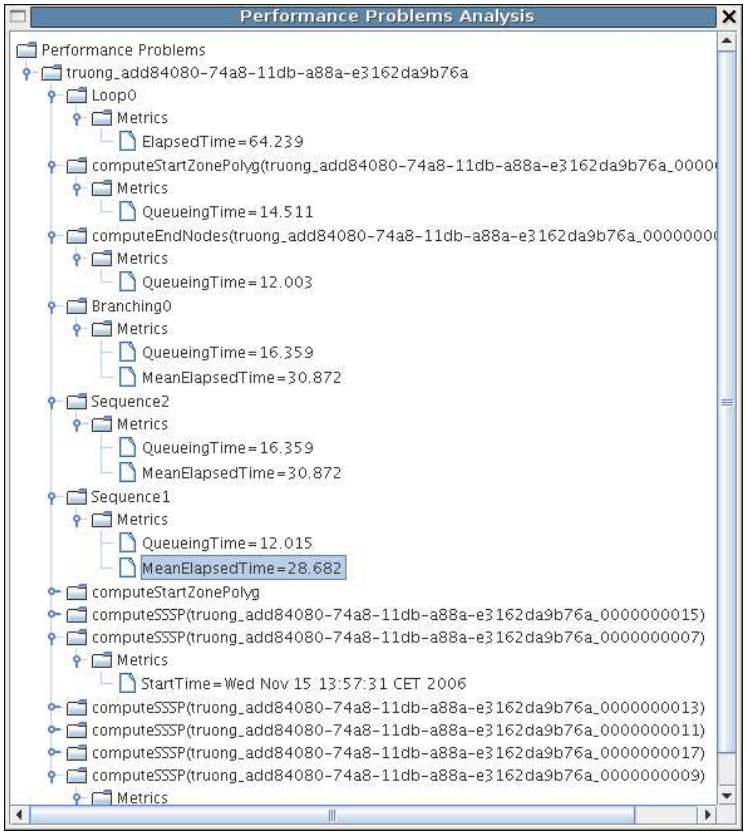

Fig. 14. Example of performance problems detected in the CTM workflow. Nodes under the tree root represent instances of workflow, workflow regions and activities. Under a node representing an instance is a subtree visualizing performance metrics.

cations. In contrast to K-WfGrid, it does not support cyclic and Web service-based workflows. The Autopilot [37] is a distributed performance measurement and resource control system that supports the monitoring and adaptation of distributed applications. G-PM/OCM-G supports the online analysis of MPI-based applications in the Grid [38]. Both Autopilot and G-PM/OCM-G do not support Grid workflows composed from Grid/Web services.

Taverna Workbench [39] supports Grid/Web service-based workflows and it also monitors activity/workflow statuses, such as running or completed. Only limited performance statistics information is provided and Taverna does not provide performance overhead analysis and search features. The OntoGrid project [40] uses knowledge gained from monitoring data to debug workflows, but not to monitor and analyze the performance. Supporting performance analysis of workflow regions, workflow overhead analysis, and search for performance problems is missing in most existing workflow analysis tools.

In our previous work in SCALEA-G, we provided a performance monitoring and visualization for ASKALON workflows [41]. SCALEA-G and DI-PAS share many monitoring and analysis methods and concepts in common.

However, performance analysis techniques in ASKALON support workflows of C/Fortran-based scientific applications. DIPAS also supports performance search based on client's requests while SCALEA-G limits to performance monitoring and overhead analysis. Also DIPAS provides a Web portal for conducting performance monitoring and analysis.

Recently, researchers have focused on workflow provenance [42,43]. Provenance data is an important source for better understanding the performance behavior of Grid workflows. Our performance analysis service could be adapted to work with other underlying workflow provenance/monitoring systems.

8 Conclusions and Future Work

In this paper, we presented DIPAS which supports online performance analysis of Grid workflows. The main contribution of the paper is a novel distributed performance analysis service that supports both workflow users (end-users and developers) and Grid workflow middleware services to analyze performance of running and completed workflows, to search for performance problems through the specification of conditions in advance or on-demand, and to provide open interfaces for Grid middleware to access performance data in order to optimize the workflow construction and execution. We have illustrated how DIPAS can help the user and Grid middleware to carry out the performance analysis through a real-world application in the K-WfGrid testbed.

Currently, we assume that the improvement of the performance of workflows is done by the user and the Grid middleware, thus it is up to them to optimize workflows. In this work we have not reported the scalability of DIPAS that will be conducted in the future work. Although designed for the K-WfGrid workflow system, the performance framework presented in this paper is interoperable and extensible, and can easily be adapted to any other workflow systems, e.g, those supporting BPEL [44]. To achieve this, DIPAS has to be able to process different workflow models. One solution is to conduct the performance analysis based on an intermediate representation of workflow models, instead of specific workflow models. To simplify the development of new analysis features and to easily extend the analysis component, our future work is to use rule-based systems for checking performance problems, instead of using customized code at the moment. We will deal with the performance of invoked applications within activities by analyzing performance data collected from Web service containers and from the instrumentation of invoked applications. Moreover, we plan to use WS-Notification for informing clients about performance problems. Further information about DIPAS can be found at `http://www.dps.uibk.ac.at/projects/kwfgrid`.

Acknowledgments

This paper refines and extends a previous paper published in [10]. We are grateful to the support of our colleagues in the K-WfGrid consortium. We thank Simon Ostermann for the implementation of the workflow region analysis. The work described in this paper is partially supported by the European Union through the IST-2002-511385 K-WfGrid project. We thank anonymous reviewers for their useful comments.

References

[1] J. Yu, R. Buyya, A taxonomy of workflow management systems for grid computing, Journal of Grid Computing 3 (3-4) (2005) 171–200.
URL http://dx.doi.org/10.1007/s10723-005-9010-8

[2] D. Kyriazis, K. Tserpes, A. Menychtas, A. Litke, T. Varvarigou, An innovative workflow mapping mechanism for grids in the frame of quality of service, Future Gener. Comput. Syst. 24 (6) (2008) 498–511.

[3] Z. Shi, J. Dongarra, Scheduling workflow applications on processors with different capabilities, Future Generation Comp. Syst. 22 (6) (2006) 665–675.

[4] J. Yu, R. Buyya, C.-K. Tham, Cost-based scheduling of scientific workflow application on utility grids., in: e-Science, IEEE Computer Society, 2005, pp. 140–147.

[5] L. Huang, D. W. Walker, O. F. Rana, Y. Huang, Dynamicworkflow management using performance data, ccgrid 0 (2006) 154–157.

[6] T. Glatard, J. Montagnat, D. Emsellem, D. Lingrand, A Service-Oriented Architecture enabling dynamic services grouping for optimizing distributed workflows execution, Future Generation Computer Systems 24 (7) (2008) 720–730.
URL http://dx.doi.org/10.1016/j.future.2008.02.011

[7] T. Heinis, C. Pautasso, G. Alonso, Design and evaluation of an autonomic workflow engine., in: ICAC, IEEE Computer Society, 2005, pp. 27–38.

[8] G. Singh, C. Kesselman, E. Deelman, Optimizing grid-based workflow execution, Journal of Grid Computing 3 (3-4) (2005) 201–219.
URL http://dx.doi.org/10.1007/s10723-005-9011-7

[9] The K-WfGrid Project. http://www.kwfgrid.eu, last access: 06 June, 2007.

[10] H.-L. Truong, P. Brunner, T. Fahringer, F. Nerieri, R. Samborski, B. Balis, M. Bubak, K. Rozkwitalski, K-WfGrid Distributed Monitoring and Performance Analysis Services for Workflows in the Grid, in: 2nd IEEE

International Conference on e-Science and Grid Computing, IEEE Computer Society, Amsterdam, The Netherlands, 2006.

[11] M. Bubak, S. Unger (Eds.), K-WfGrid - The Knowledge-based Workflow System for Grid Applications, Academic Computer Centre CYFRONET AGH, 2007, http://www.cyf-kr.edu.pl/cgw06/info/K-WfGrid_PROCEEDINGS.pdf.

[12] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, T. Storey, W. Vambenepe, S. Weerawarana, Modeling Stateful Resources with Web Services, Specification, Globus Alliance, Argonne National Laboratory, IBM, USC ISI, Hewlett-Packard (Jan. 2004).

[13] K-WfGrid GEMINI, http://gemini.icsr.agh.edu.pl/, last access: 06 June, 2007.

[14] K-WfGrid GWES (Grid Workflow Execution Service), http://www.gridworkflow.org/kwfgrid/gwes/docs/, last access: 5 June, 2007.

[15] B. Kryza, R. Slota, M. Majewska, J. Pieczykolan, J. Kitowski, Grid organizational memory - provision of a high-level grid abstraction layer supported by ontology alignment, Future Generation Comp. Syst. 23 (3) (2007) 348–358.

[16] F. Nerieri, R. Prodan, T. Fahringer, H.-L. Truong, Performance Analysis of Grid Workflow Applications, in: Proceedings of The 7th IEEE/ACM International Conference on Grid Computing (Grid'06), IEEE Computer Society Press, 2006.

[17] OpenDHT, http://www.opendht.org, last access: 06 June, 2007.

[18] K-WfGrid GOM (Grid Organizational Memory), http://gom.kwfgrid.net/web/space/welcome, last access: 06 June, 2007.

[19] The Grid Workflow Description Language (GWorkflowDL), http://www.gridworkflow.org/kwfgrid/gworkflowdl/docs/.

[20] H.-L. Truong, S. Dustdar, T. Fahringer, Performance metrics and ontologies for grid workflows, Future Gener. Comput. Syst. 23 (6) (2007) 760–772.

[21] Workflow Patterns, http://is.tm.tue.nl/research/patterns/patterns.htm, last access: 21 August 2008.

[22] Globus Project, http://www.globus.org.

[23] GridSphere Portal Framework, http://www.gridsphere.org, last access: 06 June, 2007.

[24] JFreeChart, http://www.jfree.org/jfreechart/, last access: 06 June, 2007.

[25] JGraph, http://www.jgraph.com/, last access: 06 June, 2007.

[26] The K-WfGrid Project: Coordinated Traffic Management, http://www.kwfgrid.eu/index.php?option=com_content&task=view&id=32&itemid=51, last access: 6 June, 2007.

[27] A. Knüpfer, H. Brunst, W. E. Nagel, High performance event trace visualization., in: PDP, IEEE Computer Society, 2005, pp. 258–263.

[28] H. W. Cain, B. P. Miller, B. J. N. Wylie, A callgraph-based search strategy for automated performance diagnosis (distinguished paper)., in: A. Bode, T. L. 0002, W. Karl, R. Wismüller (Eds.), Euro-Par, Vol. 1900 of Lecture Notes in Computer Science, Springer, 2000, pp. 108–122.

[29] C. Seragiotto, T. Fahringer, Performance analysis for distributed and parallel java programs with aksum., in: CCGRID, IEEE Computer Society, 2005, pp. 1024–1031.

[30] K.-H. Kim, C. A. Ellis, Performance analytic models and analyses for workflow architectures, Information Systems Frontiers 3 (3) (2001) 339–355.

[31] J. Cardoso, A. P. Sheth, J. A. Miller, Workflow quality of service., in: K. Kosanke, R. Jochem, J. G. Nell, A. O. Bas (Eds.), ICEIMT, Vol. 236 of IFIP Conference Proceedings, Kluwer, 2002, pp. 303–311.

[32] A. F. Abate, A. Esposito, N. Grieco, G. Nota, Workflow performance evaluation through wpql, in: Proceedings of the 14th international conference on Software engineering and knowledge engineering, ACM Press, 2002, pp. 489–495.

[33] WebLogic Process Integrator Overview, http://edocs.beasys.com/wlpi/wlpi11/studio/index.htm, last access: 06 June, 2007.

[34] W. D. Pauw, M. Lei, E. Pring, L. Villard, M. Arnold, J. F. Morar, Web services navigator: Visualizing the execution of web services., IBM Systems Journal 44 (4) (2005) 821–846.

[35] M. W. Johnson, Monitoring and diagnosing applications with arm 4.0., in: Int. CMG Conference, Computer Measurement Group, 2004, pp. 473–484.

[36] P. Kacsuk, G. Dozsa, J. Kovacs, R. Lovas, N. Podhorszki, Z. Balaton, G. Gombas, P-GRADE: a Grid Programming Environment, Journal of Grid Computing 1 (2) (2003) 171–197.

[37] R. L. Ribler, J. S. Vetter, H. Simitci, D. A. Reed, Autopilot: Adaptive control of distributed applications, in: HPDC, 1998, pp. 172–179.

[38] R. Wismüller, M. Bubak, W. Funika, High-level application-specific performance analysis using the g-pm tool, Future Generation Comp. Syst. 24 (2) (2008) 121–132.

[39] Taverna, http://taverna.sourceforge.net/, last access: 06 June, 2007.

[40] Ontogrid project, http://www.ontogrid.net, last access: 06 June, 2007.

[41] P. Brunner, H. L. Truong, T. Fahringer, Performance monitoring and visualization of grid scientific workflows in askalon., in: M. Gerndt, D. Kranzlmüller (Eds.), HPCC, Vol. 4208 of Lecture Notes in Computer Science, Springer, 2006, pp. 170–179.

[42] R. Bose, I. Foster, L. Moreau, Report on the international provenance and annotation workshop: (ipaw'06) 3-5 may 2006, chicago, SIGMOD Rec. 35 (3) (2006) 51–53.

[43] Y. L. Simmhan, B. Plale, D. Gannon, A framework for collecting provenance in data-centric scientific workflows, icws 0 (2006) 427–436.

[44] Business Process Execution Language for Web Services, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, last access: 06 June, 2007.