



PERFORMANCE ANALYSIS SERVICE DEVELOPER MANUAL

WP3

Document Filename: **KWF-WP3-UIBK-v0.6-PASDeveloperManual**

Work package: **WP3**

Partner(s): **UIBK, CYFRONET, FIRST**

Lead Partner: **UIBK**

Document classification: **PUBLIC**

Abstract: This development manual describes in detail the architecture and implementation of the performance analysis module (DIPAS) in the K-WfGrid project.

Delivery Slip

	Name	Partner	Date	Signature
From	Hong-Linh Truong	UIBK	04/01/2006	
Verified by	Piotr Nowakowski	CYFRONET	07/01/2006	
Approved by	Steffen Unger	FIRST	10/01/2006	

Document Log

Version	Date	Summary of changes	Author
0.1	10/11/2005	Initial text, table of content	Peter Brunner, Francesco Nerieri, Robert Ramborski, Hong-Linh Truong
0.2	30/11/2005	Change all sections	Hong-Linh Truong
0.3	01/12/2005	Change all sections	Hong-Linh Truong, with input from Peter Brunner, Francesco Nerieri, Robert Ramborski
0.4	04/01/2006	Revise all sections	Hong-Linh Truong
0.5	25/08/2006	Revise all sections	Peter Brunner, Hong-Linh Truong
0.6	31/08/2006	Revise all sections	Hong-Linh Truong
	23/09/2006	Revise all sections based on internal review comments	Hong-Linh Truong
1.0	07/10/2006	QA Check	Piotr Nowakowski
1.1	14/10/2006	Add performance search subsection	Vlad Nae
	31/10/2006	Revise all sections	Hong-Linh Truong

CONTENTS

1. COPYRIGHT NOTICE.....	4
2. INTRODUCTION.....	5
2.1. ABBREVIATIONS AND ACRONYMS.....	6
2.2. REFERENCES AND SOURCE CODE.....	6
3. IMPLEMENTATION STRUCTURE	8
3.1. USE CASES	8
3.1.1. <i>Infrastructure monitoring</i>	8
3.1.2. <i>On-line performance monitoring and analysis of workflows</i>	8
3.1.3. <i>Overhead analysis and performance problem search use case</i>	9
3.2. PROTOTYPE COMPONENT MODEL	10
3.3. DETAILED IMPLEMENTATION MODEL	10
3.3.1. <i>Infrastructure monitoring</i>	10
3.3.1.1. DIPASConfiguration	11
3.3.1.2. PDQSInput	11
3.3.1.3. DataOutput	11
3.3.1.4. DIPASPortalService.....	11
3.3.1.5. DQSDisplay	11
3.3.2. <i>Performance monitoring and analysis of workflow</i>	12
3.3.2.1. WfPMAGUIApplet	12
3.3.2.2. WfPMAGUIClient	12
3.3.2.3. ActivityExecutionTimeLine	12
3.3.2.4. KWorkflowVisualize.....	13
3.3.2.5. MonitoringService.....	13
3.3.3. <i>DIPAS Gateway</i>	13
3.3.3.1. DIPAS Gateway Factory	13
3.3.3.2. DIPASService	13
3.3.3.3. AnalysisRequestMessage	15
3.3.3.4. Sending analysis requests and receiving results.	15
3.4. PERFORMANCE PROBLEM SEARCH.....	16
3.4.1. <i>Overview of the static structure of the subcomponents</i>	16
3.4.1.1. WfPerfComponent.....	16
3.4.1.2. PerfProblemSearch.....	17
3.4.2. <i>The dynamic behaviour of the Performance Problems Search component</i>	17
3.4.2.1. General overview.....	18
3.4.2.2. Detailed illustration of Performance Problems Search component dynamics	18
4. CODE TESTING.....	19
5. CONTACT INFORMATION AND CREDITS.....	20
6. THE GNU LICENSE AGREEMENT	21
7. REFERENCES.....	22

1. COPYRIGHT NOTICE

Copyright (c) 2004-2006 Distributed and Parallel Systems Group, University of Innsbruck. All rights reserved.

Use of this product is subject to the terms and licenses stated in the GPL license agreement. Please refer to Section 6 for details.

Java™ is a registered trademark of Sun. All rights reserved.

Applet™ is a registered trademark of Sun. All rights reserved.

Jena is copyrighted by Hewlett-Packard Development Company, LP.

JFreeChart is licensed under GPL, JFree software projects (www.jfree.org).

Gridsphere is copyrighted by Max-Planck-Gesellschaft/Max-Planck-Institut fuer Gravitationsphysik.

Castor is copyrighted by Intalio Inc., and others.

Tomcat and Xerces are licensed under the Apache License, The Apache Software Foundation.

This research is partly funded by the European Commission IST-2002-511385 Project “K-WfGrid”.

2. INTRODUCTION

DIPAS (Distributed Performance Analysis Service) controls the monitoring and instrumentation service, conducts the performance analysis of workflows at runtime and provides performance metrics proposed by the metric ontology to clients. This stable version is a collection of performance monitoring and analysis tools that can be accessed by the user through a web browser, supporting the user to login from anywhere via the Internet with minimum installation effort and from virtually any platform to conduct the performance monitoring and analysis of Grid infrastructure and workflows in a user friendly way with many options available. The tools collect the data dynamically from services running in distributed Grid sites, analyze performance and monitoring data, and visualize the performance results in the portal.

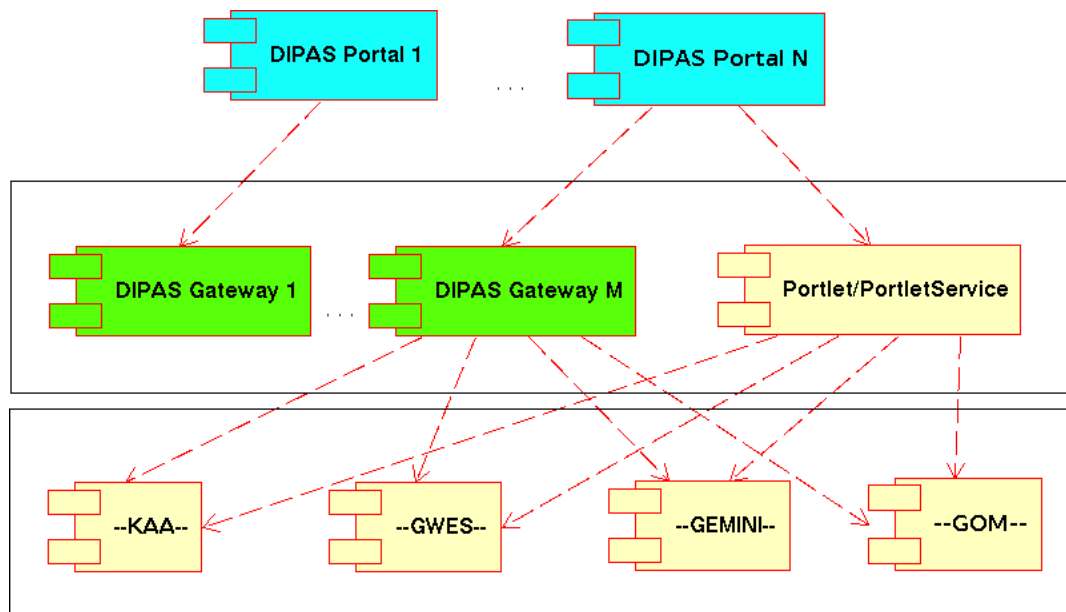


Figure 2-1: Architecture of DIPAS

Figure 2-1 depict the DIPAS which includes three main parts

- *The DIPAS portal* provides a single place for the user to conduct the performance monitoring and analysis of Grid infrastructure and workflows. The content of DIPAS portal is provided by DIPAS portlet/portletservices. Moreover, the portal also includes a Java applet implementing the main GUI of performance analysis and visualization of workflows. The applet visualizes the monitoring data of the workflows and partially analyzes the performance of the workflows. It also controls the DIPASGateways (see below) to perform the workflow overhead analysis and displays the resulting overhead analysis to the user. The Java applet is configured supporting Java Plug-in mechanism [JAVA-PLUGIN]. The portal is deployed into a web container based on Tomcat [TOMCAT].
- *The DIPAS Portlet/PortletServices:* they are implemented by using Gridsphere [GRIDSPHERE]. The portal interacts with portlets [JSR168] and portlet services which process user requests through web interfaces and generate contents displayed in the portal. Portlets/PortletServices are used to provide monitoring data of Grid infrastructure to the portal.

- *The DIPAS Gateway* is a GT4 WSRF [WSRF] service that acts as a mediator between the portal and various services (e.g., GOM, GEMINI, GWES) involved in the performance monitoring and analysis. Moreover, it implements the performance overhead analysis and search for performance problems.

2.1. ABBREVIATIONS AND ACRONYMS

Abbreviation	Description
DIPAS	Distributed Performance Analysis Service
DR	Performance Service Interfaces and Data Representation
GOM	Grid Organizational Memory
GWES	Grid Workflow Execution Service
GWUI	Grid Workflow User Interface
KAA	Knowledge Assimilation Agent
PDQS	Performance Data Query and Subscription
URL	Uniform Resource Locator
WARL	Workflow Analysis Request Language
WF	Workflow
WfID	Workflow ID
WfMS	Workflow Management System
WSRF	Web Service Resource Framework

2.2. REFERENCES AND SOURCE CODE

The whole DIPAS source code consists of two parts:

- `dipasportal`: including DIPAS Portal and portlets/portletservices
- `dipasservice`: including DIPAS Gateway

All the source code can be downloaded from K-WfGrid CVS. Figure 2-2 presents the structure of source directory of `dipasportal`. There are four main directories:

- The `lib` directory contains all the third party libraries on which the portal is dependent. DIPASPortal is dependent on several third party libraries like Castor [CASTOR], JFreeChart [JFREECHART], JGraph [JGRAPH] the K-WfGrid DR package [DRUSER][DRDEV], Junit[JUNIT], SCALEA-G [SCALEAG] and other XML Apache libraries [XMLAPACHE].
- The `src` directory includes all Java source files developed in DIPASPortal.
- The `webapp` directory includes HTML and JSP files of DIPASPortal and signed JAR files for the applet.
- The `test` directory includes JUnit test files

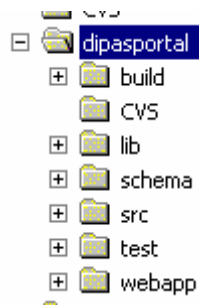


Figure 2-2: Source code directory structure of dipasportal.

Figure 2-3 presents the structure of source directory of dipasservice. There are five main directories:

- The `lib` directory includes the third party libraries. Important libraries here are Castor, Glassbox, K-WfGrid Gworkflowdl, GWUI, JFreechart, JUnit, the K-WfGrid DR package, and the SCALEA-G libraries.
- The `samples` directory consists of XML samples.
- The `schema` contains WSDL files of Grid services of DIPASGateway.
- The `src` directory contains Java source files of DIPASGateway.
- The `test` directory includes JUnit test files

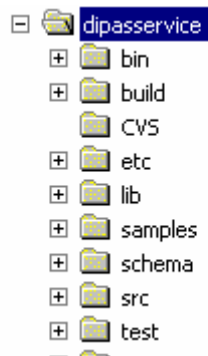


Figure 2-3: Source code directory structure of dipasservice.

3. IMPLEMENTATION STRUCTURE

3.1. USE CASES

We defined three main use cases: the first use case describes the infrastructure monitoring, the second one explains the workflow monitoring and analysis, and the third one describes the overhead analysis and the search for performance problems for workflows.

3.1.1. Infrastructure monitoring

In this use case we describe the monitoring of Grid infrastructure monitoring that can be conducted by using DIPASPortal.

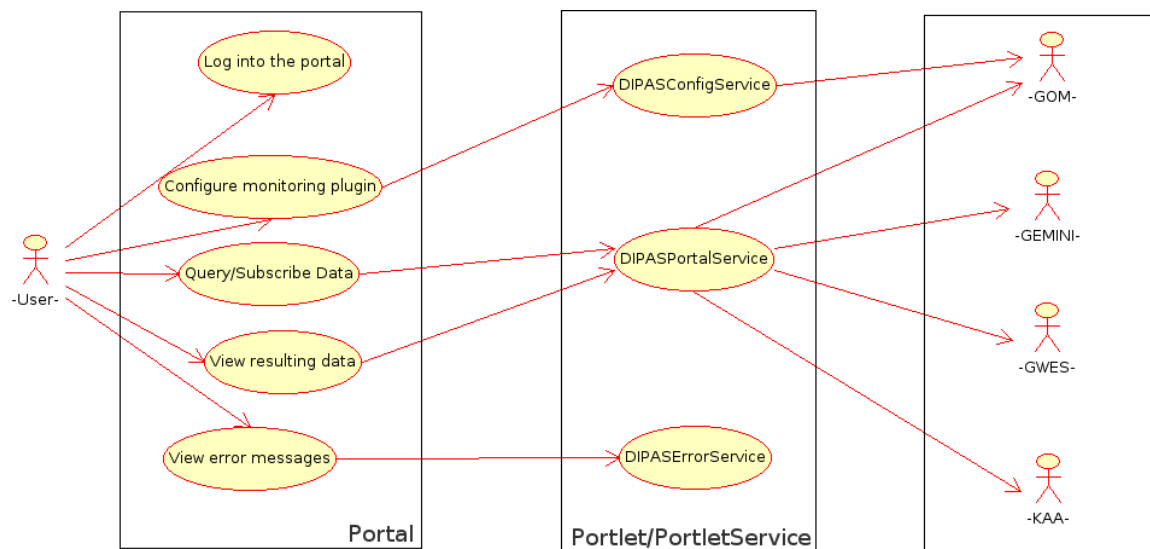


Figure 3-1: Infrastructure monitoring use case.

The web portal is the main interface through which the user conducts the monitoring of the Grid infrastructure. After logging into the portal, the user can select the monitoring service plug-in to be used; the portal can be configured to use different monitoring services, named GEMINI [GEMINI] and SCALEA-G [SCALEA-G]. Then, the user retrieves all the available data that can be queried or subscribed. The user can choose the data of interest and query or subscribe the data. The subscribed and/or queried data can be examined on the fly. If there are any errors, they can be viewed by the user.

The portal interacts with different services, using Portlets and PortletServices; these services run in the background and retrieve the monitoring information from the monitoring service chosen. The Portlets are used to get the input from the user, process the input and output the data when the result comes back from the different services like GOM, GEMINI, etc. PortletServices are used to store the requests and the output of a user so that the user can access this data later on, after logging out from the system. When logging into the system, the user can access the data that has been processed previously.

3.1.2. On-line performance monitoring and analysis of workflows

In this use case we describe the performance monitoring and analysis of an existing workflow that is conducted through DIPASPortal.

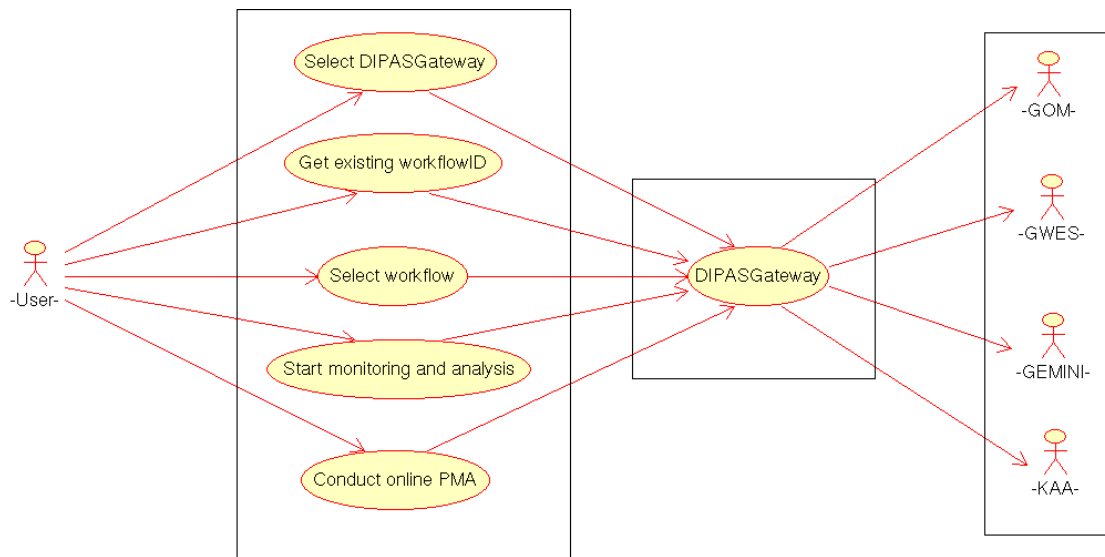


Figure 3-3: Use case for online performance monitoring and analysis of workflows.

After connecting and logging in to the portal, the user can start the performance monitoring and analysis of workflows. Firstly, the user requests DIPASGateway to provide the existing workflowIDs. Secondly, after getting existing workflowIDs, the user can select one of the existing workflows. Then, the user can start the monitoring of the workflow. The states of the workflow activities will be updated in the graph when the execution state changes. The workflow description will be retrieved and the workflow is displayed as a graph. During the execution of the workflow, the user can conduct an on-line performance and monitoring analysis. To do all these actions, the portal will communicate with the DIPASGateway, which interacts with the other services involving in the performance monitoring and analysis.

3.1.3. Overhead analysis and performance problem search use case

Figure 3-4 presents a use case of overhead analysis and search for performance problems. Through the portal a client prepares a WARL performance request and sends the request to the Performance Search Component. The resulting overhead will be computed based on the Overhead Analysis and the graph of the workflow. The Overhead Analysis retrieves the monitoring data from GEMINI whereas the workflow description of the workflow is obtained from GWES. The performance result will be described using XML representation, defined in K-WfGrid DR module, and returned to the portal.

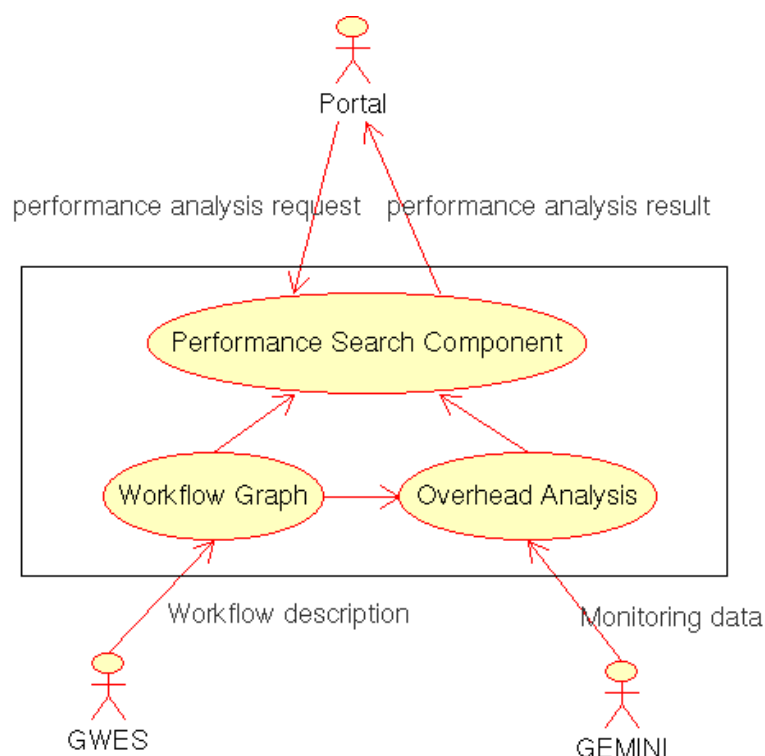


Figure 3-4: Overhead analysis and performance problem search use case.

3.2. PROTOTYPE COMPONENT MODEL

Figure 2-1 presents the system components of DIPAS and how they interact with each other. In our component model we can have several DIPAS Portals that communicate with several DIPAS Gateways for the performance monitoring and analysis of the workflow execution. Every DIPAS Portal can communicate with the Portlets and the PortletServices for the infrastructure monitoring. When the user logs in to the portal, the user can access the Portlet/PortletServices, as we described in the first use case (infrastructure monitoring) or the user can use the DIPASGateway, as we described in the second use case (online performance monitoring and analysis of workflow).

3.3. DETAILED IMPLEMENTATION MODEL

In this section, we describe in detail the most important classes of DIPAS components of the implementation of our stable version. For how to configure, deploy and use DIPAS components, reader should refer to DIPAS user manual [DIPASUSER].

3.3.1. Infrastructure monitoring

Infrastructure monitoring is conducted through a DIPAS Portal that supports the user to query and subscribe interesting monitoring data of Grid infrastructure. The portal is developed based on Gridsphere [GRIDSPHERE] which is an open-source framework for developing portlet-based Web portal for the Grid. Gridsphere has to be built with Ant, and the portal can be deployed in Apache Tomcat container [TOMCAT]. We have implemented Portlets and PortletServices to access the different services like GOM, GEMINI and GWES that are involved in the performance monitoring and analysis.

3.3.1.1. *DIPASConfiguration*

DIPASConfiguration class is an extension of *ActionPortlet* class. This class is used to set different configurations, based on that the portlets/portlet services know which monitoring plug-in it should use. The portlets/portlet services need to know the underlying monitoring system that they use. Information about underlying monitoring systems is specified by monitoring-plugin name and registry information of the monitoring system.

The source file of *DIPASConfiguration* is stored under `dipas/dipasportal/src/net/kwfgrid/dipas/portlets/DIPASConfiguration.java` in KWfGrid CVS.

3.3.1.2. *PDQSInput*

PDQSInput is a portlet that processes most interactions/requests given by the user through the portal. It is an extension of the *ActionPortlet* class. After a valid PDQS request specified, the request will be processed by *PDQSInput* which in turn sends the request to the monitoring service. The resulting data is processed and displayed by the *DataOutput* class.

The source file of *PDQSInput* is stored under `dipas/dipasportal/src/net/kwfgrid/dipas/portlets/PDQSInput.java`.

3.3.1.3. *DataOutput*

DataOutput is the portlet that displays the resulting monitoring data in the portal. It is an extension of the *ActionPortlet* class. The data can be displayed in two modes: static and dynamic. Static data is displayed in text mode and dynamic data is displayed in figures which are updated over the time. We will explain display modes in detail in Section 3.3.1.5. The figures are composed in other classes, but in *DataOutput* these pictures are rearranged, each associated with a corresponding request.

The source file of *DataOutput* is stored under `dipas/dipasportal/src/net/kwfgrid/dipas/portlets/DataOutput.java`.

3.3.1.4. *DIPASPortalService*

DIPASPortalService is a *PortletService* that is used to store the requests of the different users. It is a container that, once created, is stored in the session of the user. Thus, if a user logs out, the data remains persistent and if the user logs in on another machine the user can see the results that the user requested. Another function of *DIPASPortalService* is used to update available data given by the monitoring service; the update is periodically conducted within an interval of a predefined time. Therefore, it makes sure that the user has always an up-to-date list of services that the user can query or subscribe.

The source file of *DIPASPortalService* is stored under `dipas/dipasportal/src/net/kwfgrid/dipas/portlets/services/DIPASPortalService.java` and `dipas/dipasportal/src/net/kwfgrid/dipas/portlets/services/impl/DIPASPortalServiceImpl.java`.

3.3.1.5. *DQSDisplay*

The portal has to visualize the different types of data that requires different customized displays. In order to make the visualization extensible and easily to be configured, we use an abstract class called *DQSDisplay*; all types of displays implement interfaces provided by this class. Therefore, the

developer can easily add a new display for a new data type of the infrastructure data. The data can be displayed in two modes: dynamic mode and static mode. Figure 3-5 presents existing visualizations for different types of monitoring data.

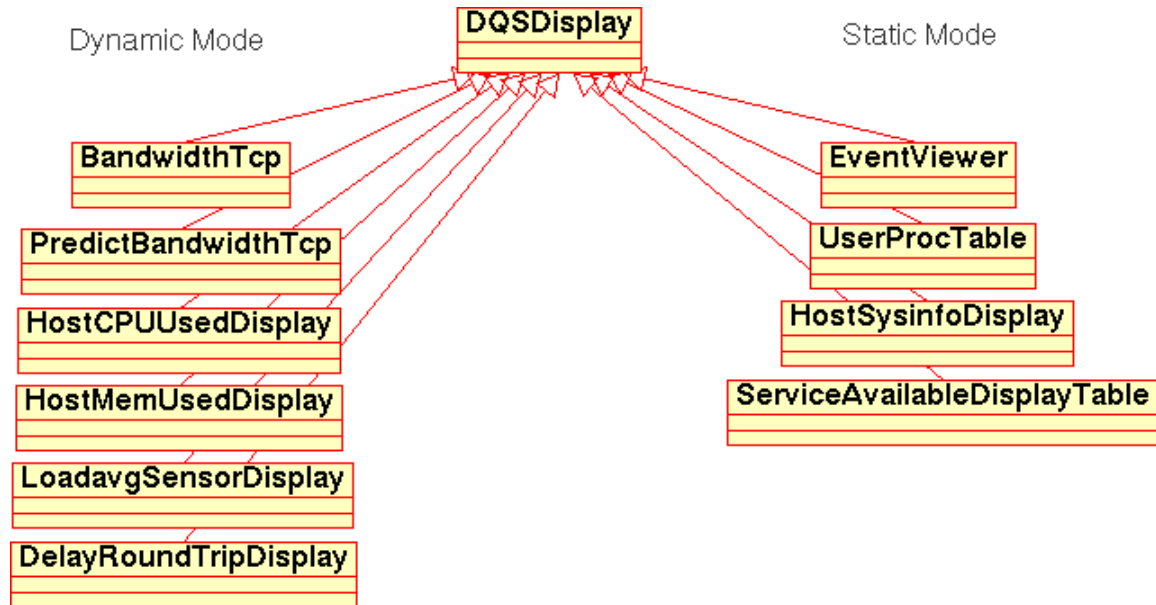


Figure 3-5: Class diagram of DQSDisplay

3.3.2. Performance monitoring and analysis of workflow

In this section, we describe in detail the most important components in the performance monitoring and analysis of workflow of our DIPASPortal. This part of the portal is made with Java Applet technology. Besides the Web portal for conducting performance monitoring and analysis of workflows, a Java standalone application with similar features is also provided.

3.3.2.1. WfPMAGUIApplet

WfPMAGUIApplet is an extension of JApplet class. It implements an applet that partially conducts performance analysis, visualizes workflow performance data and is responsible for the whole communication between the user and the DIPASGateway.

The source of WfPMAGUIApplet is stored under `dipas/dipasservice/src/net/kwfguid/dipas/client/WfPMAGUIApplet.java`.

3.3.2.2. WfPMAGUIClient

WfPMAGUIClient implements a Java standalone application whose function is similar to WfPMAGUIApplet.

The source of WfPMAGUIClient is stored under `dipas/dipasservice/src/net/kwfguid/dipas/client/WfPMAGUIClient.java`.

3.3.2.3. ActivityExecutionTimeLine

ActivityExecutionTimeLine is a JPanel that displays the performance monitoring results of the workflow. The source of ActivityExecutionTimeLine is stored under `dipas/dipasservice/src/net/kwfguid/dipas/up/ActivityExecutionTimeLine.java`.

3.3.2.4. *KWorkflowVisualize*

The `KWorkflowVisualize` class implements the visualization of workflows. It is used to visualize the workflow graph. This class can visualize workflows described by K-WfGrid `GWorkflowDL` (<http://www.gridworkflow.org/kwfgrid/gworkflowdl/docs/>), and the visualization is based on K-WfGrid `GWUI` package.

The source of `KWorkflowVisualize` is stored under `dipas/dipasservice/src/net/kwfgrid/dipas/up/KWorkflowVisualize.java`.

3.3.2.5. *MonitoringService*

The interface `MonitoringService` provides abstract interfaces for implementing different plugins to different monitoring services. There are two implementations of this interface, one for the monitoring service `GEMINI`, the other for the monitoring service `SCALEA-G`.

The source file of `MonitoringService` is stored under `dipas/dipasservice/src/net/kwfgrid/dipas/ms/MonitoringService.java`

3.3.3. *DIPAS Gateway*

`DIPAS Gateway` is a `GT4.0 WSRF` service. `DIPAS Gateway` is implemented using factory model:

- Firstly, clients connect to `DIPASFactory`, creating a `WSRF` resource and retrieving the `EPR` (endpoint reference) of the new-created resource.
- Secondly, clients send requests (for example, subscribing and querying data and searching for performance problems) by invoking corresponding operations on the resource.

Clients can send analysis requests, described in `WARL`, to search for performance problems. These requests are then processed by the `Performance Search Component` which is a part of `DIPAS Gateway`. During runtime, the client/user can get performance results. The `Performance Search Component` is intended to analyze the performance of `Grid workflows` and `Grid applications`, identifying performance overheads of workflows at activity and workflow levels and detecting performance bottlenecks.

3.3.3.1. *DIPAS Gateway Factory*

The factory provides only a service operation as follows

```
<operation name="CreatedDIPAS">
    <input message="tns:CreatedDIPASRequest" />
    <output message="tns:CreatedDIPASResponse" />
</operation>
```

(see the `WDSL` file of `DIPASFactory` in `dipas/dipasservice/schema/serviceinterface/DIPAS/DIPASFactoryService.wsdl`)

This operation is used to create `WSRF` resources managed by the `DIPAS Gateway` service.

3.3.3.2. *DIPASService*

The `DIPASService` implements service operations that manipulate `WSRF` resource. Its main service operations are

- `public boolean ping(net.kwfgrid.dipas.serviceinterface.Ping parameters)` throws `java.rmi.RemoteException`: implements service-level ping operation.
- `public java.lang.String analyze(net.kwfgrid.dipas.message.AnalysisRequestMessage request)` throws `java.rmi.RemoteException`: accepts performance data query and subscription requests, and analysis requests. It then invokes corresponding services to process the requests. The resulting data will be buffered at DIPAS Gateway before retrieved by the requesters. This operation returns a result ID associating with the resulting data.
- `public net.kwfgrid.dipas.serviceinterface.DataEntries getResult(java.lang.String resultID)` throws `java.rmi.RemoteException`: returns the resulting data of a request indicated by the given `resultID`.
- `public net.kwfgrid.dipas.serviceinterface.CancelResultResponse cancelResult(java.lang.String resultID)` throws `java.rmi.RemoteException`: cancels receiving resulting data of a request.
- `public net.kwfgrid.dipas.serviceinterface.GetWorkflowIDResponse getWorkflowIDs(net.kwfgrid.dipas.serviceinterface.GetWorkflowID request)` throws `java.rmi.RemoteException`: returns a list of existing workflow IDs
- `public java.lang.String getWorkflowDescription(java.lang.String workflowID)` throws `java.rmi.RemoteException`: returns the workflow description of the workflow indicated by the input `workflowID`.
- `public net.kwfgrid.dipas.serviceinterface.StopPMAResponse stopPMA(java.lang.String workflowID)` throws `java.rmi.RemoteException`: stop all analysis tasks associated with the given `workflowID`.
- `public net.kwfgrid.dipas.serviceinterface.SuspendWorkflowResponse suspendWorkflow(java.lang.String workflowID)` throws `java.rmi.RemoteException`: suspend the workflow currently being monitored and analyzed. The workflow is identified by the parameter `workflowID`.
- `public net.kwfgrid.dipas.serviceinterface.ResumeWorkflowResponse resumeWorkflow(java.lang.String workflowID)` throws `java.rmi.RemoteException`: resume the monitored and analyzed workflow which is currently being suspended. The workflow is identified by the parameter `workflowID`.

When using DIPASGateway to conduct the performance analysis.

- Firstly, invoke `getWorkflowIDs` service operation to obtain existing workflow IDs
- Secondly, select a `workflowID` and conduct an analysis task by
 - Specify an analysis request typed `AnalysisRequestMessage`,
 - Invoke `analyze()` service operation with the analysis request and get a `resultID`, and
 - Poll the resulting data by invoking `getResult()` with the corresponding `resultID`

3.3.3.3. *AnalysisRequestMessage*

AnalysisRequestMessage class can be used to specify different requests, e.g. PDQS and WARL requests using the same representation. The main content of *AnalysisRequestMessage* schema is shown below:

```
<xsd:complexType name="AnalysisRequestMessage">
  <xsd:sequence>
    <xsd:element name="content" type="xsd:string" />
    <xsd:element name="language" type="xsd:string" /
    <xsd:element name="command" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

- `content`: the content of the request. The representation of the content is dependent on the language of the request
- `language`: the language of the request. The language could be PDQL or WARL
- `command`: the command that applies to the request. For example, command could be: to query or subscribe monitoring data, or to analyze workflow overheads

For example, if a client wants to *subscribe* all events of a workflow with ID= `truong_96eeb880-125c-11db-a105-ce90a766c196`, then the client can specify the following *AnalysisRequestMessage*:

```
String pdqsRequest = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<PDQS><dataTypeID>wfa.event</dataTypeID><resourceID>truong_96eeb
880-125c-11db-a105-
ce90a766c196</resourceID><subscriptionTime><from>0</from><to>0</
to></subscriptionTime></PDQS>" ;
```

```
AnalysisRequestMessage message = new AnalysisRequestMessage();
//PDQS language
message.setLanguage("PDQS");
//request content
message.setContent(pdqsRequest);
//subscribe data
message.setCommand("COMMAND_SUBSCRIBE");
```

3.3.3.4. *Sending analysis requests and receiving results.*

The following code shows an example of how to send an request and receive the result.

```
AnalysisRequestMessage message = new AnalysisRequestMessage();
message.setLanguage("PDQS");
message.setContent(RequestHelper.createPDQSString("wfa.event", wfID
, status));
message.setCommand("COMMAND_SUBSCRIBE");
final String resultID = resource.analyze(message);
//if result is not null, we have data
if (resultID!=null) {
    //update the current monitored workflow
    ...
    boolean poll =true ;
    while (poll) {
        DataEntries entries= resource.getResult(resultID);
        //no more data
        if (entries==null) {
            poll =false ;
            cancel();// end Timer if no more data
        }
        String [] results=entries.getEntry();
        for (int i=0; i <results.length;i++) {
            DataEntries entries= resource.getResult(resultID);
            ...
        }
    }
}
```

3.4. PERFORMANCE PROBLEM SEARCH

This component is running on the service-side and its task is to analyze the performance of the workflows (running or completed) and to report performance problems if any. It is started when the user generates requests through the DIPAS Portal or the Java stand-alone client. Its tasks include computing basic and extended metrics both for activities and for workflow-regions and reporting back only if there are performance problems. The component is also used for monitoring workflows, in which case it computes all the requested metrics and returns the results.

3.4.1. Overview of the static structure of the subcomponents

The Performance Problem Search component has two subcomponents: an extension of the `java.lang.Thread` class, `WfPerfComponent`, and an analysis subcomponent, `PerfProblemSearch`. A `WfPerfComponent` thread is created for each monitored workflow for which the user requests an analysis. Each thread uses one `PerfProblemSearch` subcomponent. The role of the threads is mainly dispatching requests and events and the role of the `PerfProblemSearch` subcomponents is the actual computation of metrics and analysis of requests (Figure 3-4-1).

3.4.1.1. *WfPerfComponent*

As mentioned before, the main task that `WfPerfComponent` has is the dispatching of activity events and messages. This is achieved using a few methods:

- `postCreate(String workflowId)` – creates a new instance of the `PerfProblemSearch` subcomponent for the specified workflow

- `addEntry(String entry) / addEntry(String[] entries)` – adds events to the event queue to be passed to the `PerfProblemSearch` for analyzing.
- `sendWarl(String warlString, String resultUUID)` – forwards the WARL requests received from the client-side to the `PerfProblemSearch`
- `getPerfResults(String warlString, String resultUUID)` – forwards the ApprofXML result returned by the `PerfProblemSearch`, to the client
- `stopThread()` – stops the thread which is running for managing the event queue. This method is called when the user chooses to stop monitoring the performance of the corresponding workflow.

3.4.1.2. *PerfProblemSearch*

This subcomponent does the actual metric computing and the problem analysis. The basic behaviour is: the subcomponent receives the events sent by `WfPerfComponent` and computes/updates the requested metrics if necessary. The results are retrieved by pooling, using the `getPerfResults()` method.

- `analyzePerf(String warlRequest, String resultUUID)` – receives a WARL request from the client, analyzes, parses it and adds it to a internal list of performance problem requests.
- `updateOn(String entityId)` – this method is called when the monitoring service generates an event related to an activity or the workflow. The `entityId` represents either the id of an activity instance, either the workflow id, depending on which one generated the event. This method is doing the actual computing of the metrics and the analyzing of the requests.
- `getPerfResults(String warlRequest, String resultUUID)` – returns the existing computed results if there are any, in approfXML format.

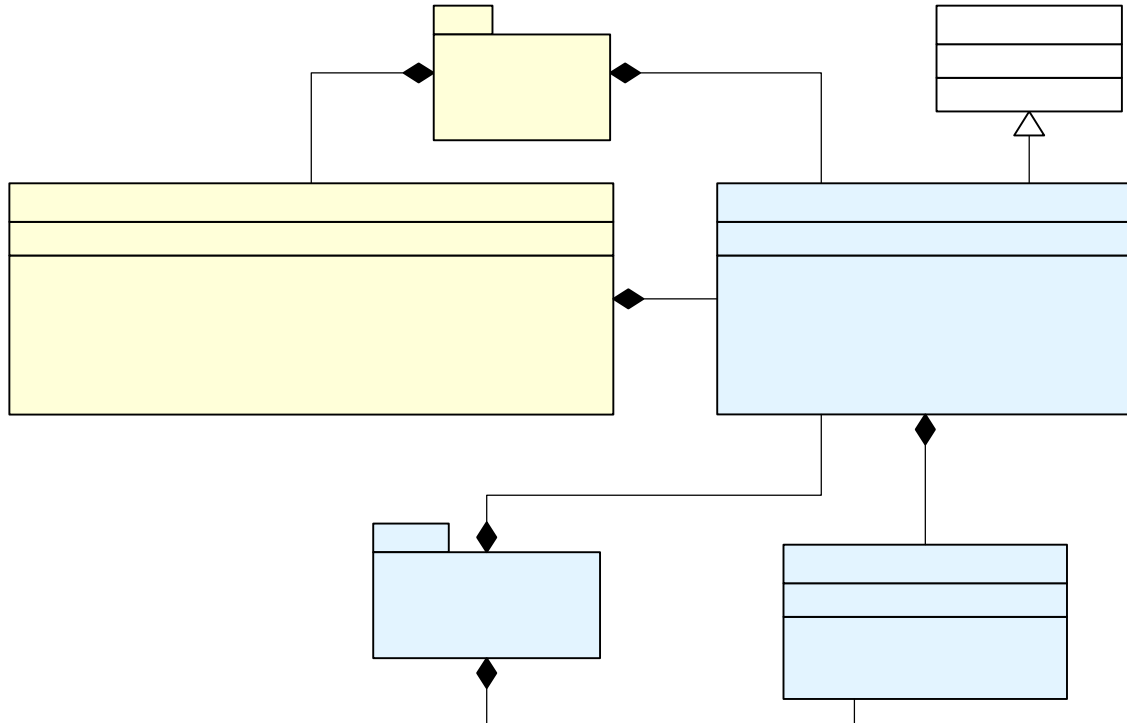


Figure 3-4-1. Static structure of Performance Problems Search component

3.4.2. The dynamic behaviour of the Performance Problems Search component

3.4.2.1. General overview

The basic dynamics of the Performance Problems Search component are:

- Each workflow is associated with a component
- The events are distributed between components, when dispatching, each event being forwarded to its corresponding component
- Also the requests coming from various clients are dispatched to their corresponding component
- The metrics are dynamically computed and if there are problems (according to the requests) results are being generated
- The clients have to pool the service in order to get the available results.
- All client–service communication is realized evidently by XML messaging, the performance problem analysis requests being sent in WARL-XML format and the results being sent back in appropXML format.

3.4.2.2. Detailed illustration of Performance Problems Search component dynamics

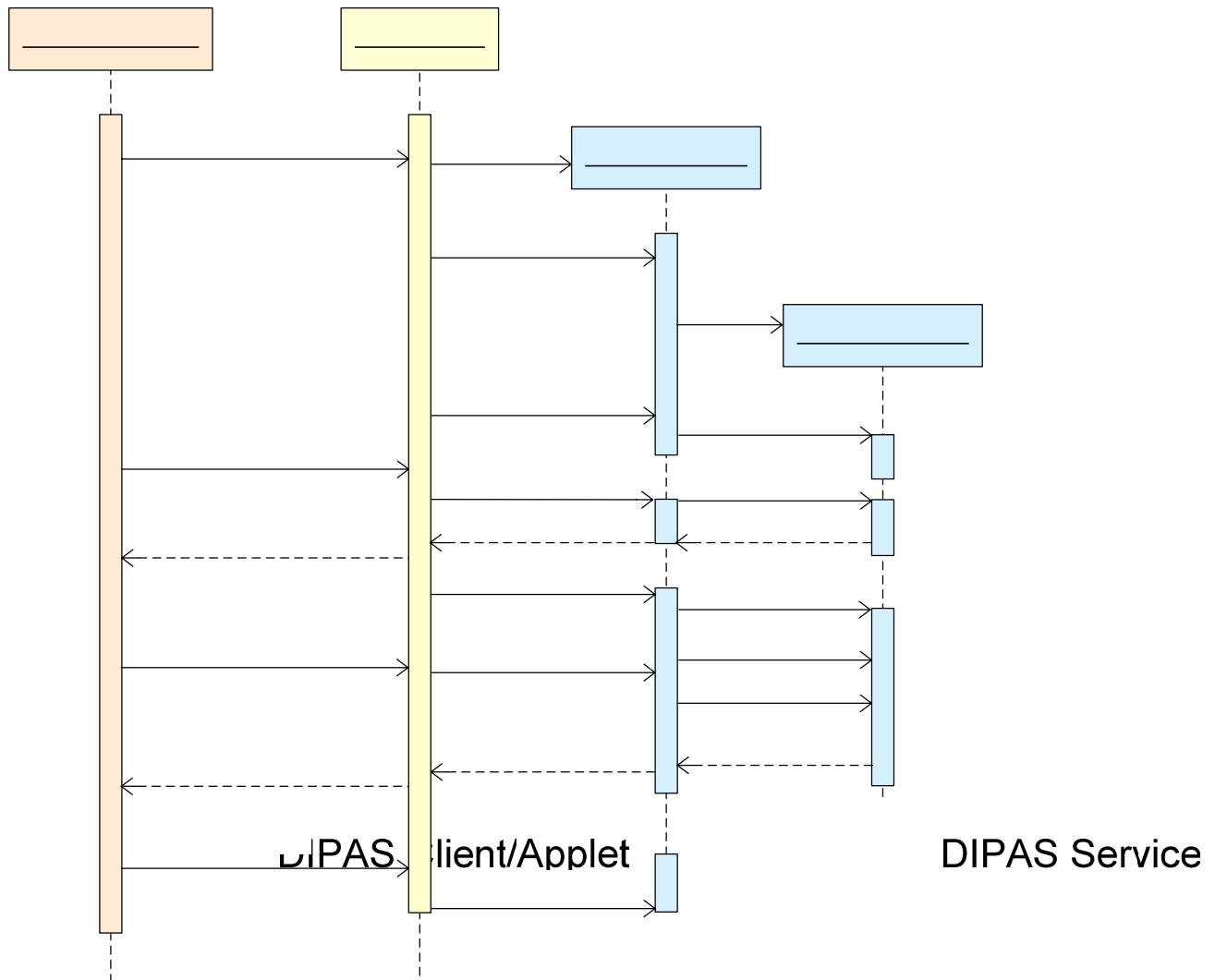


Figure 3-4-2-2. Simplified sequence diagram illustrating the dynamics of the Performance Problems Search component

4. CODE TESTING

For testing DIPASPortal and DIPASGateway which are written in Java, we partially use JUnit to test our implementation. However, for visualization of monitoring results we cannot use JUnit because we cannot check the result of visualization. For all the Java source code programs, we use Log4J (<http://logging.apache.org/log4j/docs/>) for logging.

5. CONTACT INFORMATION AND CREDITS

DIPAS framework is developed at:

Distributed and Parallel Systems Group,
Institute of Computer Science, University of Innsbruck
Technikerstrasse 21A, A-6020, Innsbruck, Austria

The developers include:

- Peter Brunner
- Vlad Nae
- Robert Samborski
- Hong-Linh Truong (team leader)

With contributions from:

- Francesco Nerieri, Simon Ostermann

Visit <http://www.dps.uibk.ac.at/projects/pma> for further information about our work on performance monitoring and analysis. Further comments, suggestions, and bug reporting, contact Hong-Linh Truong at truong@dps.uibk.ac.at.

6. THE GNU LICENSE AGREEMENT

Copyright (c) 2004-2006 Distributed and Parallel Systems Group, University of Innsbruck. All rights reserved.

This software includes voluntary contributions made to K-WfGrid. For more information on K-WfGrid, please see <http://www.kwfgrid.eu>.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7. REFERENCES

- [APPLET] Sun Developer Network, <http://java.sun.com/applets/>
- [CASTOR] <http://www.castor.org/>
- [DIPASUSER] K-WfGrid Consortium, "K-WfGrid Performance Analysis Service -User Manual", August, 2006.
- [DRDEV] K-WfGrid Consortium, "K-WfGrid Performance Service Interfaces and Data Representation: Development Manual", August, 2006
- [DRUSER] K-WfGrid Consortium, "K-WfGrid Performance Service Interfaces and Data Representation: User Manual", August, 2006
- [JAVA PLUG-IN] Sun Developer Network, <http://java.sun.com/j2se/1.4.2/docs/guide/plugin/index.html>
- [JFREECHART] JFreeChart, <http://www.jfree.org/jfreechart/>.
- [JGRAPH] JGraph, <http://www.jgraph.com>.
- [JSR168] JSR 168: Portlet Specification. <http://www.jcp.org/en/jsr/detail?id=168>
- [JUNIT] <http://www.junit.org/>
- [GEMINI] B. Balis, M. Bubak, J. Dziwisz, H.-L. Truong, and T. Fahringer, "Integrated Monitoring Framework for Grid Infrastructure and Applications", In P. Cunningham and M. Cunningham, editors, Innovation and the Knowledge Economy. Issues, Applications, Case Studies, pages 269-276, Ljubljana, Slovenia, October 2005. IOS Press.
- [GLOBUS] Globus Toolkit. <http://www-unix.globus.org/toolkit/>
- [GRIDSPHERE] The Gridsphere Portal Framework. <http://www.gridisphere.org/gridisphere/gridisphere>
- [SCALEA-G] Hong-Linh Truong and Thomas Fahringer. "SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid", Scientific Programming, 12(4):225-237, IOS Press, 2004.
- [WPPERFONTO] Hong-Linh Truong, Thomas Fahringer, Francesco Nerieri, Schahram Dustdar "Performance Metrics and Ontology for Describing Performance Data of Grid Workflows, IEEE International Symposium on Cluster Computing and Grid 2005 (CCGrid2005), 1st International Workshop on Grid Performability, IEEE Computer Society Press, Cardiff, UK, 9 - 12 May 2005.
- [WSRF] Ian Foster et al., "Modeling Stateful Resources with Web Services", Globus Alliance, Argonne National Laboratory, IBM, USC ISI, Hewlett-Packard, Jan, 2004. <http://www.globus.org/wsrp/ModelingState.pdf>.
- [TOMCAT] The Apache Software Foundation, <http://tomcat.apache.org/>
- [XERCES] <http://xerces.apache.org/>
- [XMLAPACHE] <http://xml.apache.org/>