



SCHEDULER DEVELOPER MANUAL

WP2

Document Filename:	KWF-WP2-D2-UIBK-v1.0-SchedulerDeveloperManual.doc
Work package:	WP2
Partner(s):	UIBK
Lead Partner:	UIBK
Document classification:	PUBLIC

Abstract: This document is a developer manual of the K-WfGrid Scheduler which is a service responsible for scheduling of scientific workflows based on Petri Net representation. The manual describes the implementation details of the Scheduler, including the five available scheduling algorithms – two basic algorithms and three advanced algorithms based on performance data. It also describes from the developer's perspective the service interfaces – a Web Service interface, a Java API interface, and a Java servlet interface. The document describes also JUnit tests used to test the Scheduler functionalities.

Delivery Slip

	Name	Partner	Date	Signature
From	Marek Wieczorek	UIBK	20/09/2006	
Verified by	Piotr Nowakowski	CYFRONET	08/10/2006	
Approved by	Steffen Unger	FIRST	08/10/2006	

Document Log

Version	Date	Summary of changes	Author
0.1	1/12/2005	First version	Marek Wieczorek
0.2	29/08/2006	Manual for the final version of the product.	Marek Wieczorek
0.3	20/09/2006	Corrections after an internal review.	Marek Wieczorek
1.0	08/10/2006	QA check	Piotr Nowakowski

CONTENTS

COPYRIGHT NOTICE	4
1. INTRODUCTION.....	5
1.1. ABBREVIATIONS AND ACRONYMS	6
1.2. REFERENCES AND SOURCE CODE	6
2. DESCRIPTION OF THE PRODUCT	7
3. IMPLEMENTATION STRUCTURE	8
3.1. PROTOTYPE USE CASES	8
3.2. PROTOTYPE COMPONENT MODEL	8
3.3. DETAILED IMPLEMENTATION MODEL	9
3.3.1.1. DETAILED DESCRIPTION OF THE SCHEDULING ALGORITHMS	10
3.4. PRODUCT INTERFACES	14
4. PRODUCT TESTING	16
4.1. SCHEDULER TEST CASES	16
4.1.1. <i>GWorkflowDLConverterTest</i>	16
4.1.2. <i>SchedulerTest</i>	16
4.1.3. <i>EasySchedulingTest</i>	16
4.1.4. <i>RandomTest</i>	16
4.1.5. <i>PerformanceDrivenSchedulingTest</i>	16
4.1.6. <i>KAAProxyTest</i>	16
4.1.7. <i>KAABasedSchedulingTest</i>	16
5. CONTACT INFORMATION AND CREDITS.....	18
6. THE EDG LICENSE AGREEMENT	19

COPYRIGHT NOTICE

Copyright (c) 2005 by *UIBK*. All rights reserved.

Use of this product is subject to the terms and licenses stated in the EDG license agreement. Please refer to Section 5 for details.

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

1. INTRODUCTION

The K-WfGrid project provides a complete application execution and composition environment for the Grid. The application model adopted in the environment is the scientific workflow based on the Petri Net notation which is a successful representation used to describe the behavior of distributed systems. The workflow description language used in K-WfGrid is called GWorkflowDL. Workflows can be described on different abstraction levels (represented by different colors), of which only the lowest *green* level allows for execution. The Web Service technology is used in K-WfGrid as the representation describing activity interfaces. On the lowest abstraction level, every workflow activity (every workflow *transition*, using the Petri Net notation) is described by a web service interface, as a web service operation. For processing and execution of the workflows is responsible a service called Grid Workflow Execution Service (GWES). GWES can process the workflows which are on the lowest abstraction level. For transformation between different levels of abstraction are responsible other services of the K-WfGrid, called Workflow Conversion Tool (WCT), Automatic Application Builder (AAB), and the Scheduler. Dependences between different components of the K-WfGrid environment are depicted in Figure 1.

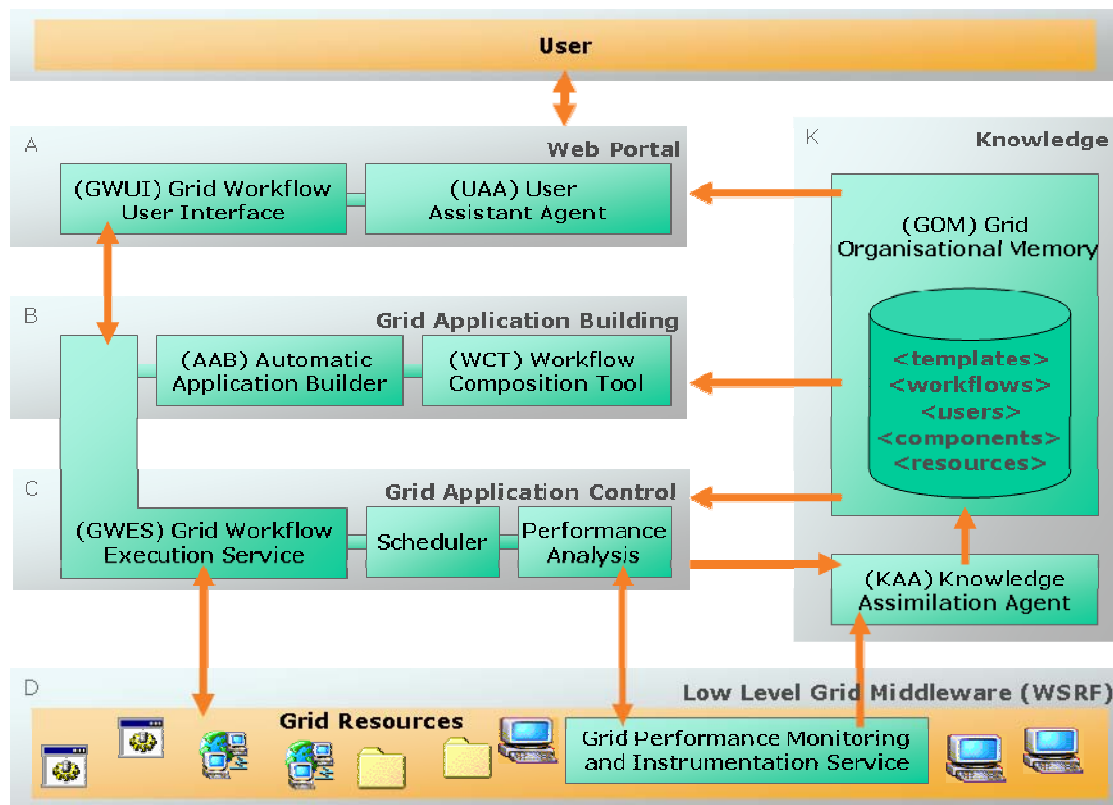


Figure 1 K-WGrid environment

The Scheduler is a component of the Grid Application Control layer. It determines which candidate web service operations (hereafter referred to also as “service instances”) are selected in order to be used in the current workflow execution. The goal is to consider non-functional properties of the web services to choose between functionally equivalent instances of the same services, and to make the workflow ready for execution. Using the K-WfGrid terminology, the Scheduler converts so-called *blue* transitions (which are assigned to alternative *service candidates*) to *green* transitions (assigned to single *service instances*).

Performance-based dynamic scheduling is a challenging problem for the Grid, due to the distribution, dynamicity and unreliability of the environment. The Scheduler tries to take advantage of both performance measurements provided by the Monitoring Service (WP3) and performance predictions provided by KAA (WP5), in order to make the most optimal scheduling decision. The Scheduler uses this information and applies a flexible scheduling strategy depending on the data which is available.

1.1. ABBREVIATIONS AND ACRONYMS

AAB – Automatic Application Builder

GWES – Grid Workflow Execution Service

GWorkflowDL – Grid Workflow Description Language

HEFT – Heterogeneous Earliest Finish Time

KAA – Knowledge Assimilation Agent

WCT – Workflow Composition Tool

1.2. REFERENCES AND SOURCE CODE

Further online information about the Scheduler is available at the following links:

- Scheduler Java Docs: <http://www.dps.uibk.ac.at/~marek/kwfgrid/scheduler/docs/>
- Scheduler CVS (password protected): <http://cvs.ui.sav.sk/cgi-bin/cvsweb.cgi/kwfgrid/scheduler/>

2. DESCRIPTION OF THE PRODUCT

Performance-oriented dynamic workflow scheduling is a challenging problem. It consists in assigning workflow activities to Grid resources over time, trying to optimize single or multiple objective functions (usually, execution time). In the existing research, the problem has been addressed in different ways. Workflows have been scheduled either statically (full-ahead planning), or fully dynamically at the execution time (just-in-time mapping). Also some intermediate solutions have been applied, for instance a so-called “workflow partitioning”, in which workflow partitions, to which a workflow is partitioned at the beginning of execution, are scheduled in a full-ahead manner, just before their execution begins. Different scheduling algorithms have been used, belonging to three main groups of algorithms: list scheduling algorithms, clustering algorithms, and duplication-based algorithms. In the list scheduling algorithms, workflow activities are scheduled in the order determined by a specially created list. The clustering algorithms include a special pre-processing phase, in which the workflow activities are grouped into special clusters (usually based on the criterion of the inter-activity communication). In the duplication-based algorithms, the activities are simply mapped to multiple resources, in order to increase the probability of optimal mapping. Some other non-standard approaches have also been applied, including Genetic Algorithms and other heuristics.

The K-WfGrid Scheduler is an integral part of the K-WfGrid environment, and is designed to provide effective scheduling methods, with use of the tools developed within the project. The Scheduler recognizes the special workflow representation format used in the project (based on colored Petri Nets), and applies different scheduling algorithms, among them an extension of the Heterogeneous Earliest Finish Time (HEFT) algorithm (see Section 3.3.1.1). For each workflow transition which can be scheduled (i.e., for each *blue* transition), the scheduler selects one of the alternative service instance candidates assigned to the transitions, which service instance will be subsequently used in the execution. The scheduling decisions are supported by two types of data: performance monitoring of resources, provided by the Monitoring Service, and knowledge-based execution time predictions, provided by the KAA. Effective combination of these two types of data for the scheduling purposes was the most important goal of the Scheduler.

3. IMPLEMENTATION STRUCTURE

3.1. PROTOTYPE USE CASES

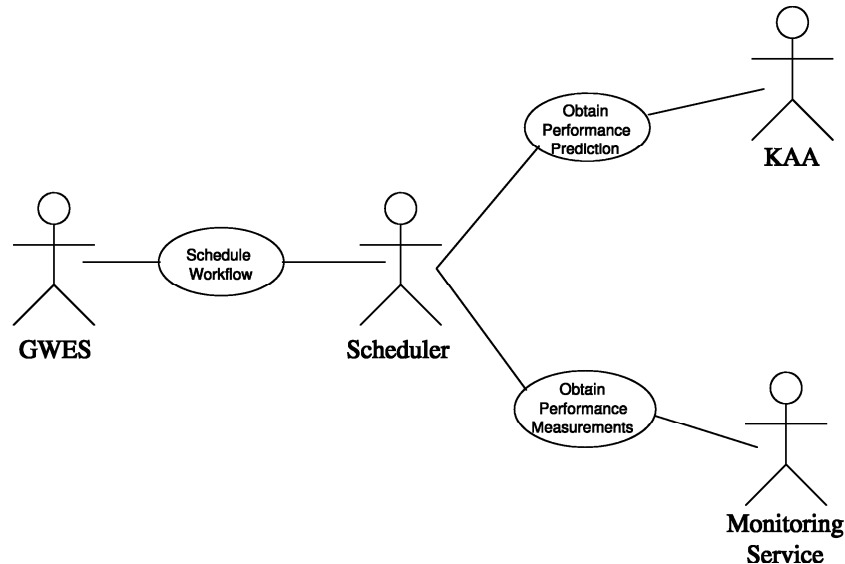


Figure 2 Use Case diagram of the Scheduler

The Scheduler is a service used by GWES each time when a workflow produced by AAB contains *blue* transitions (assigned to alternative *service candidates*). The Scheduler tries to make best possible scheduling, getting additional information from auxiliary services. Performance predictions which allow estimation of expected execution time for different service instances can be obtained from KAA (WP5). Performance measurements (both static and dynamic information about the resources) can be obtained from the Monitoring Service (WP3). If the auxiliary data is not available for some resources or deployments, the Scheduler uses default values.

3.2. PROTOTYPE COMPONENT MODEL

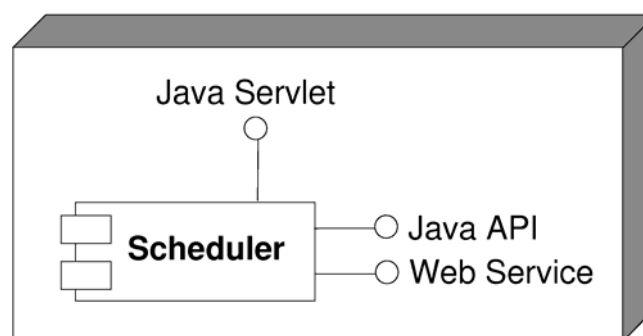


Figure 3 Scheduler component diagram

The Scheduler consists of a single software component (see Figure 3), which can be accessed by the user via different interfaces. It can be deployed as an Axis web service and accessed via a web service interface, or used as a normal Java library and accessed via Java API. The user can configure and monitor the deployed service using a graphical web interface (a Java servlet).

3.3. DETAILED IMPLEMENTATION MODEL

Source code of the deliverable is accessible in the CVS repository:

repository root: `:pserver:<username>@cvs.ui.sav.sk:/home/cvs`

repository path: `kwfgrid/scheduler`

The documentation in JavaDoc is available on:

<http://dps.uibk.ac.at/~marek/kwfgrid/scheduler/docs/index.html>

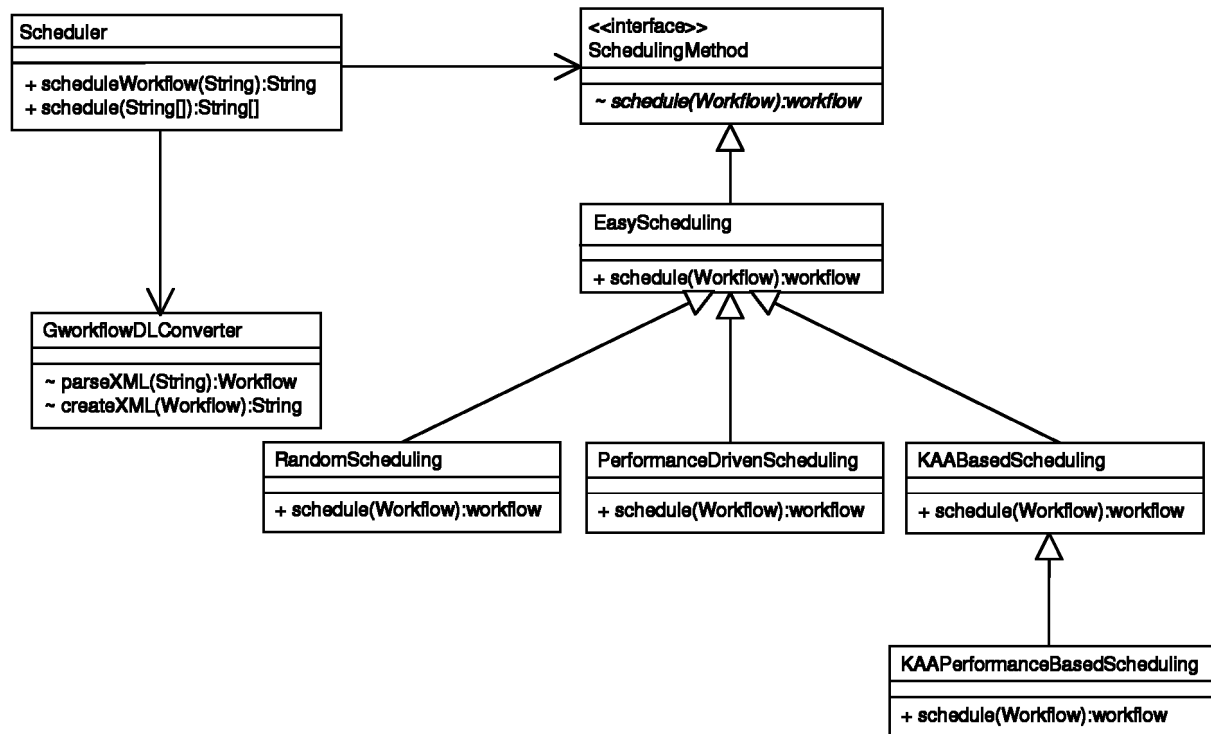


Figure 4 Class diagram of the Scheduler

The Scheduler implements several scheduling algorithms which can be used as interchangeable plugins. The *easy* algorithm (class *EasyScheduling*) provides a basic functionality of the scheduler, changing workflows from *blue* to *green* by selecting always the first option in the list of service instance candidates for each transition. The *random* algorithm (class *RandomScheduling*) is a simple modification of the *easy* algorithm, which selects service instances in a random way (rather than always the first one). The other three algorithms implement more advanced scheduling techniques, using data from other K-WfGrid services. The *performance-based* algorithm (class *PerformanceDrivenScheduling*) uses data from the Monitoring Service (WP3) to select the resources which offer the most appropriate execution conditions, from the point of view of performance. The *prediction-based* algorithm (class *KAABasedScheduling*) uses execution time predictions from the KAA service (WP5) to minimize the execution time of workflows. The *prediction-and-performance-based* algorithm (class *KAAPerformanceBasedScheduling*) combines the two aforementioned algorithms, and applies the predictions modified by the current performance information to optimize the execution time of workflows. The last two mentioned algorithms implement an extension of the Heterogeneous Earliest Finish Time (HEFT) algorithm to schedule workflows.

3.3.1.1. DETAILED DESCRIPTION OF THE SCHEDULING ALGORITHMS

Different scheduling algorithms apply different strategies to choose the service instances to be executed on the Grid. An example operation of a scheduling algorithm is depicted in Figure 5.

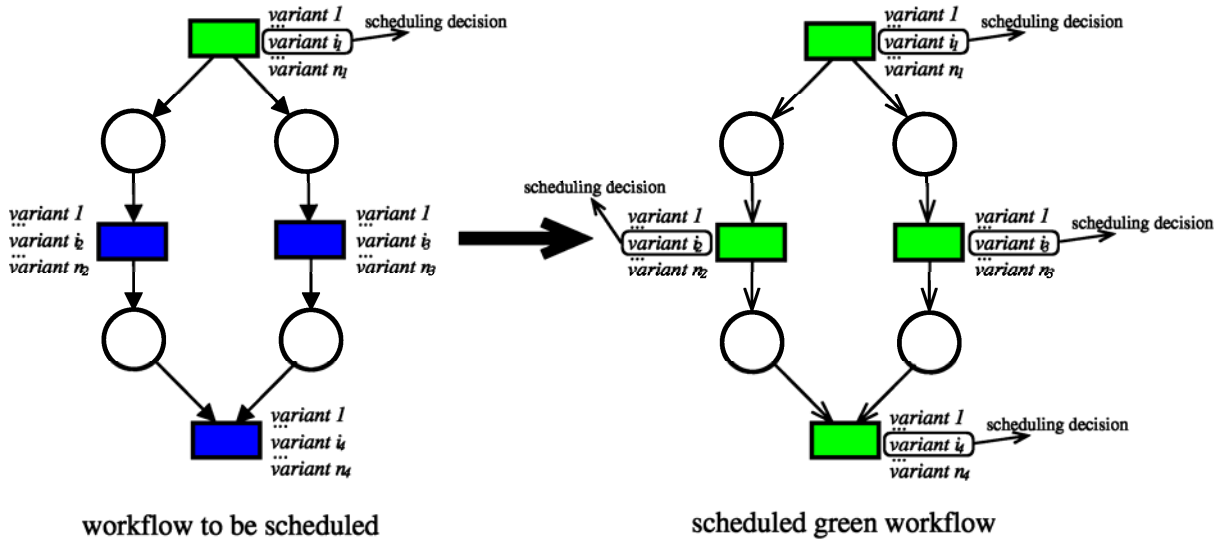


Figure 5 Example of workflow scheduling

In addition, if the *groupID consistency mode* is enabled, the Scheduler tries to fulfill a common requirement to schedule operations of the same web service to the same physical resource.

EASY SCHEDULING ALGORITHM

The *easy* scheduling algorithm (class *EasyScheduling*) traverses a whole workflow and schedules all *blue* transitions in the workflow. The algorithm chooses always the *first* variant of service instances assigned to a single transition (see Figure 6).

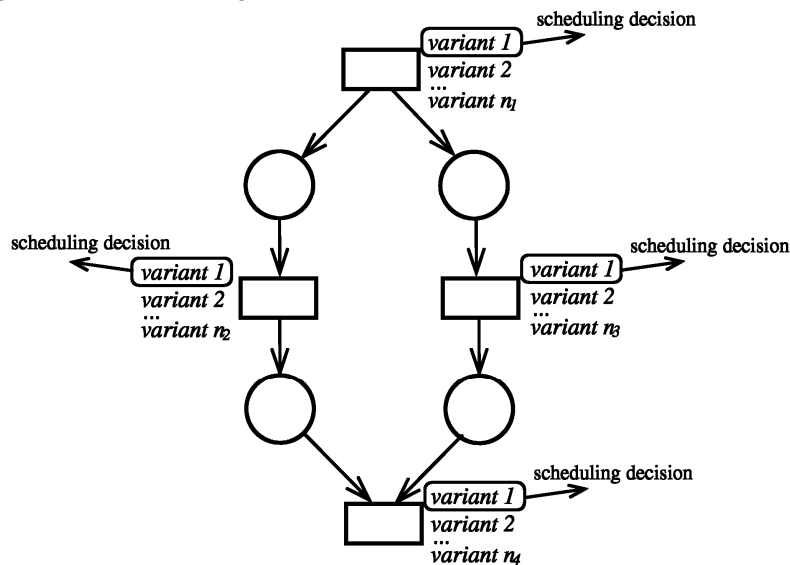


Figure 6 Example result from the easy scheduling algorithm

RANDOM SCHEDULING ALGORITHM

The *random* scheduling algorithm (class *RandomScheduling*) is similar to the random algorithm. The only difference is that it selects service instances in a random way (see Figure 7), not always the first candidate, as in case of the easy algorithm.

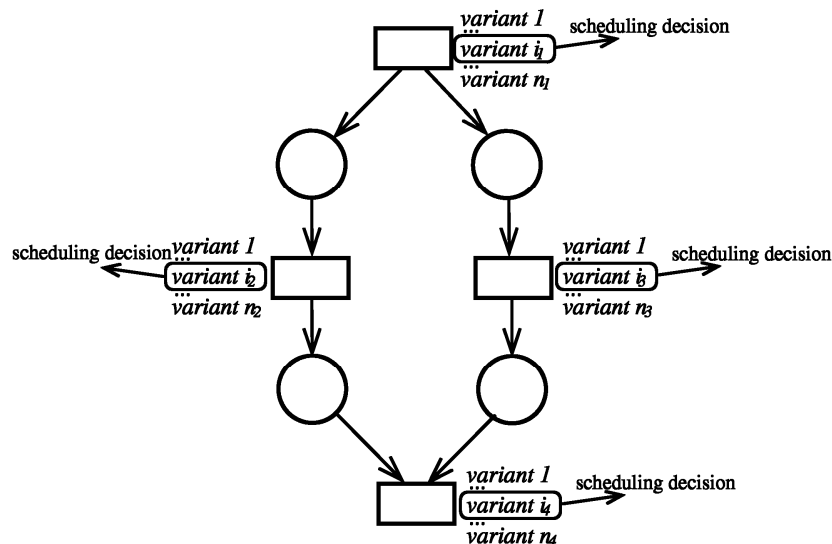


Figure 7 Example result from the random scheduling algorithm

PERFORMANCE-BASED SCHEDULING ALGORITHM

The *performance-based* algorithm is based on performance data (both static and dynamic), taken from the Monitoring Service (WP3). The Scheduler queries the Monitoring Service to get the following data:

- for each physical resource r :
 - number of CPUs of the resource: n
 - CPU speed of the processors on the resource: $cpu_speed(p)$
- for each CPU p_1, p_2, \dots, p_n , fraction of CPU which is idle (available):
 $available_cpu(p_k), available_cpu(p_k) \in (0\%, 100\%)$

The algorithm calculates a special performance indicator P for each physical resource:

$$P(r) = cpu_speed(r) \cdot \sum_{i=1}^n available_cpu(p_i) \cdot q^{i-1}$$

r – a physical resource

q – arbitrarily determined coefficient for multiple CPUs, $q \in [0, 1]$ (by default $q=1.0$)

The scheduling decisions made by the algorithm try to maximize the performance indicator, by selecting always the instance of the service which is deployed on the resource with the maximum value of P (see Figure 8).

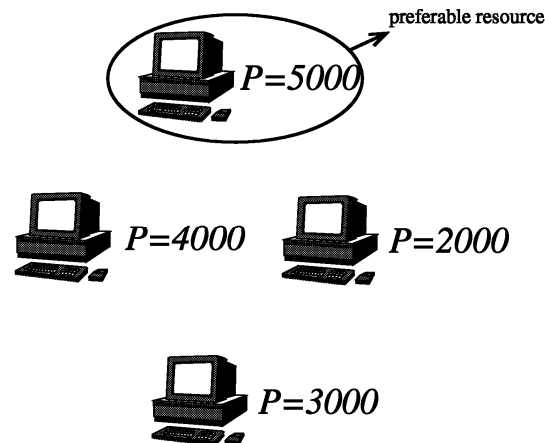


Figure 8 Performance indicators for an example Grid

PREDICTION-BASED SCHEDULING ALGORITHM

The *prediction-based* algorithm utilizes execution time predictions produced by the KAA service to minimize the execution time of workflows. The predictions for every service instance are considered in the scheduling. The algorithm extends the Heterogeneous Earliest Finish Time (HEFT), adjusting it to the workflow model based on colored Petri Nets. The algorithm consists of 3 phases:

1. Weighting phase

For each workflow transition, the Scheduler calculates the average of execution time predictions over all service instance candidates of the transition. The result is assigned to the transition as its *weight*.

2. Ranking phase

The workflow is traversed upwards, and each transition receives a *rank* value being the sum of the weight of the transition and the maximum of the ranks of its immediate successors. The rank values determine the order in which the transitions should be scheduled (the higher the rank value, the earlier is scheduled the transition).

3. Mapping phase

Following the order determined in the ranking phase, the transitions are assigned to resources by choosing the service instance which gives the minimal completion time for the given transition.

An example workflow scheduled with HEFT is shown in Figure 9.

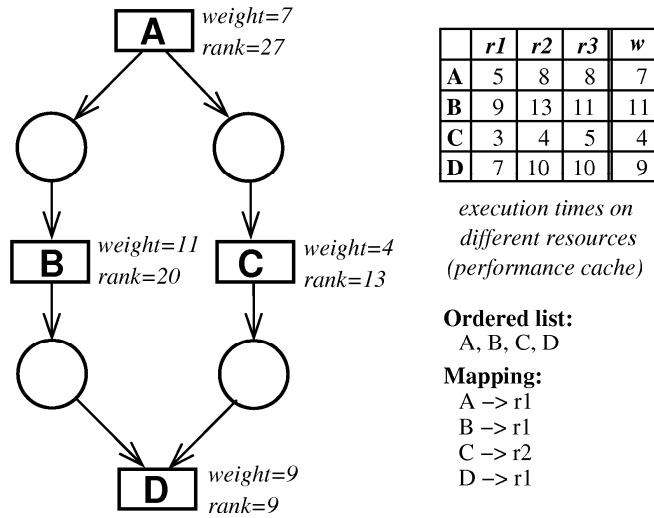


Figure 9 Example workflow scheduled with HEFT

PREDICTION-AND-PERFORMANCE-BASED SCHEDULING ALGORITHM

The *prediction-and-performance-based* algorithm combines the two algorithms mentioned above. It takes the execution time predictions from the KAA service and modifies them according to the performance data taken from the Monitoring Service. Then, it runs the algorithm described above based on the modified execution time predictions.

In order to calculate the modified execution time predictions, the Scheduler takes from the Monitoring Service the following data:

- for each physical resource r :
 - number of CPUs of the resource: n
- for each CPU p_1, p_2, \dots, p_n , fraction of CPU which is idle (available):
 $available_cpu(p_k), available_cpu(p_k) \in (0\%, 100\%)$

The algorithm calculates a special performance indicator P for each physical resource:

$$P(r) = \frac{1 - q^n}{(1 - q) \cdot \sum_{i=1}^n available_cpu(p_i) \cdot q^{i-1}}$$

r – a physical resource

q – arbitrarily determined coefficient for multiple CPUs, $q \in [0, 1]$ (by default $q=1.0$)

The modified execution time prediction is calculated in the following way:

$$METP(d) = P(resource(d)) \cdot ETP(d)$$

d – a service instance (deployment)

$ETP(.)$ – execution time prediction

$P(.)$ – performance indicator

3.4. PRODUCT INTERFACES

The Scheduler provides two main interfaces which allow the user to schedule workflows, either as an Axis web service or via Java API as a normal Java library. In both cases, we have access to the public methods of class *Scheduler*:

Scheduler Method Summary	
java.lang.String[]	<u>schedule</u> (java.lang.String[] gworkflows) Schedules an array of GWorkflowDL workflows onto available resources. The Scheduler searches inside the GWorkflowDL documents for enabled activities that are linked with Web Service operations and selects the best suited Web Service instances due to a given policy by modifying the “selected” attribute.
Java.lang.String	<u>scheduleWorkflow</u> (java.lang.String gworkflow) Schedules a GWorkflowDL workflow onto available resources. The Scheduler searches inside the GWorkflowDL document for enabled activities that are linked with Web Service operations and selects the best suited Web Service instances due to a given policy by modifying the “selected” attribute.
void	<u>setTestingMode</u> (boolean isTestingMode) Standard setter for the static field <i>isTestingMode</i> .
boolean	<u>isTestingMode</u> () Standard getter for the static field <i>isTestingMode</i> .
void	<u>setScheduledWorkflowsHistory</u> (HashMap<String, ScheduledWorkflowInfo> scheduledWorkflowsHistory) Standard setter for the static field <i>scheduledWorkflowsHistory</i>
HashMap<String, ScheduledWorkflowInfo>	<u>getScheduledWorkflowsHistory</u> () Standard getter for the static field <i>scheduledWorkflowsHistory</i>

The Scheduler Configuration Servlet (see Figure 10) is a Java servlet acting as an auxiliary interface for configuration and monitoring of the deployed scheduler service. The code of the servlet is available in class *net.kwfgrid.scheduler.servlet.SchedulerConfigurationServlet*.

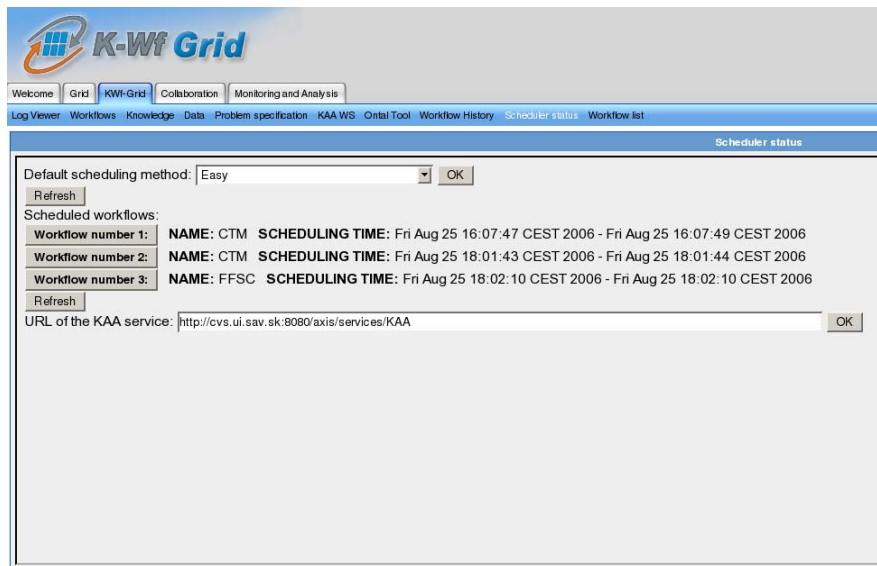


Figure 10 Scheduler Configuration Servlet

4. PRODUCT TESTING

Several JUnit tests have been used to check functionalities of the Scheduler. We checked, with respect to validity, the results returned by the Scheduler, and the partial functionalities of the Scheduler. We also checked the communication between the Scheduler and the auxiliary services (the Monitoring Service and the KAA).

4.1. SCHEDULER TEST CASES

4.1.1. GWorkflowDLConverterTest

Test the functionality of GWorkflowDL parsing

testParseXML	Tests parsing/creation of an example XML document.
--------------	----------------------------------------------------

4.1.2. SchedulerTest

Tests the Scheduler

testCtmAabOutputWf	Test the Scheduler using a simple CTM workflow created by AAB.
--------------------	----------------------------------------------------------------

4.1.3. EasySchedulingTest

Tests the *easy* scheduling algorithm.

testSchedule	Tests the algorithm using a simple FFSC workflow.
testSchedule1	Tests the algorithm with regard to the <i>principle of groupID consistency</i> , using a special "cross" workflow.

4.1.4. RandomTest

Tests the *random* scheduling algorithm.

testSchedule	Tests the algorithm using a simple FFSC workflow.
--------------	---------------------------------------------------

4.1.5. PerformanceDrivenSchedulingTest

Tests the *performance-driven* scheduling algorithm and the communication with the Monitoring Service.

testScheduleTransition	Tests the algorithm and the communication with the Monitoring Service, using a special workflow.
------------------------	--------------------------------------------------------------------------------------------------

4.1.6. KAAProxyTest

Tests the communication with the KAA service.

testEstimateRunTime	Tests the communication with the KAA service, by querying for simple execution time predictions.
---------------------	--------------------------------------------------------------------------------------------------

4.1.7. KAABasedSchedulingTest

Tests the *prerformance-driven* scheduling algorithm and the communication with the KAA service.

testInitializeSuccessorList1	Tests a partial functionality of the algorithm - proper recognition of dependencies between BLUE transitions, using an FFSC workflow.
------------------------------	---------------------------------------------------------------------------------------------------------------------------------------

testInitializeSuccessorList2	Tests a partial functionality of the algorithm - proper recognition of dependencies between BLUE transitions, using an FFSC workflow.
testInitializeSuccessorList3	Tests a partial functionality of the algorithm - proper recognition of dependencies between BLUE transitions, using a special "cross" workflow.
testInitializeSuccessorList4	Tests a partial functionality of the algorithm - proper recognition of dependencies between BLUE transitions, using a special "cross" workflow including a loop.
testgetLastBlueTransitions1	Tests a partial functionality of the algorithm – proper recognition of "last" BLUE transitions, using a CTM workflow.
testInitializePredecessorList1	Tests a partial functionality of the algorithm – proper recognition of backward dependencies between BLUE transitions, using a special "cross" workflow.
testInitializePredecessorList2	Tests a partial functionality of the algorithm – proper recognition of backward dependencies between BLUE transitions, using a special "cross" workflow.
testRankingPhase	Tests the ranking phase of the algorithm, using a special "tree" workflow.
testRankingPhase1	Tests the ranking phase of the algorithm, using a CTM workflow.
testMappingPhase1	Tests the mapping phase of the algorithm, using a special "ring" workflow.

5. CONTACT INFORMATION AND CREDITS

Contact person: Marek Wieczorek, (email: marek.wieczorek@uibk.ac.at)

6. THE EDG LICENSE AGREEMENT

Copyright (c) 2004 K-WfGrid. All rights reserved.

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

Installation, use, reproduction, display, modification and redistribution of this software, with or without modification, in source and binary forms, are permitted. Any exercise of rights under this license by you or your sub-licensees is subject to the following conditions:

1. Redistributions of this software, with or without modification, must reproduce the above copyright notice and the above license statement as well as this list of conditions, in the software, the user documentation and any other materials provided with the software.
2. The user documentation, if any, included with a redistribution, must include the following notice: "This product includes software developed by K-WfGrid (www.kwfgrid.net).” Alternatively, if that is where third-party acknowledgments normally appear, this acknowledgment must be reproduced in the software itself.
3. The names "K-WfGrid" and "Knowledge Workflow Grid" may not be used to endorse or promote software, or products derived therefrom, except with prior written permission by steffen.unger@first.fraunhofer.de.
4. You are under no obligation to provide anyone with any bug fixes, patches, upgrades or other modifications, enhancements or derivatives of the features, functionality or performance of this software that you may develop. However, if you publish or distribute your modifications, enhancements or derivative works without contemporaneously requiring users to enter into a separate written license agreement, then you are deemed to have granted participants in K-WfGrid a worldwide, non-exclusive, royalty-free, perpetual license to install, use, reproduce, display, modify, redistribute and sub-license your modifications, enhancements or derivative works, whether in binary or source code form, under the license conditions stated in this list of conditions.

5. DISCLAIMER

THIS SOFTWARE IS PROVIDED BY K-WfGrid AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, OF SATISFACTORY QUALITY, AND FITNESS FOR A PARTICULAR PURPOSE OR USE ARE DISCLAIMED. K-WfGrid AND CONTRIBUTORS MAKE NO REPRESENTATION THAT THE SOFTWARE, MODIFICATIONS, ENHANCEMENTS OR DERIVATIVE WORKS THEREOF, WILL NOT INFRINGE ANY PATENT, COPYRIGHT, TRADE SECRET OR OTHER PROPRIETARY RIGHT.

6. LIMITATION OF LIABILITY

K-WfGrid AND CONTRIBUTORS SHALL HAVE NO LIABILITY TO LICENSEE OR OTHER PERSONS FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, EXEMPLARY, OR PUNITIVE DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, LOSS OF USE, DATA OR PROFITS, OR BUSINESS INTERRUPTION, HOWEVER CAUSED AND ON ANY THEORY OF CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE), PRODUCT LIABILITY OR OTHERWISE, ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.