



WCT USER MANUAL

WP2

Document Filename:	KWF-WP2-D2-CYF-v1.4-WCTUserManual.doc
Work package:	WP2
Partner(s):	CYFRONET
Lead Partner:	CYFRONET
Document classification:	PUBLIC

Abstract: This document forms a *live* user-targeted documentation for the K-WfGrid Workflow Composition Tool (WCT).

Delivery Slip

	Name	Partner	Date	Signature
From	Tomasz Gubała, Maciej Malawski	CYFRONET	06/10/2006	
Verified by	Piotr Nowakowski	CYFRONET	08/10/2006	
Approved by	Steffen Unger	FIRST	08/10/2006	

Document Log

Version	Date	Summary of changes	Author
1.0	01/12/2005	First version	Tomasz Gubała, Maciej Malawski
1.1	01/12/2005	Some corrections	Tomasz Gubała, Maciej Malawski
1.2	29/12/2005	Review-advised changes	Tomasz Gubała
1.3	18/08/2006	New version for final release	Daniel Haręźlak, Tomasz Gubała
1.4	19/09/2006	Version after review	Daniel Haręźlak, Tomasz Gubała

CONTENTS

COPYRIGHT NOTICE	4
1. INTRODUCTION.....	5
1.1. ABBREVIATIONS AND ACRONYMS	5
1.2. REFERENCES AND SOURCE CODE	5
2. PRODUCT USAGE	6
2.1. RUNNING THE PRODUCT	6
2.1.1. <i>Operating Requirements</i>	6
2.1.1.1. Local hardware requirements	6
2.1.1.2. Local software requirements	6
2.1.1.3. Grid infrastructure requirements	7
2.1.2. <i>Step-by-Step User Setup</i>	7
2.2. BASIC OPERATION	7
2.2.1. <i>Short introduction to initial workflows in GWorkflowDL</i>	8
2.2.2. <i>Simple workflow patterns in WCT</i>	9
2.2.3. <i>WCT Optimization algorithms</i>	10
2.2.3.1. AND-branch optimization algorithm	10
2.2.3.2. Dependency reduction algorithm	11
2.3. ADVANCED FEATURES.....	12
2.3.1. <i>Deploying WCT tool as a Web Service</i>	12
2.3.2. <i>Using WCT through a web service interface</i>	13
2.4. KNOWN PROBLEMS.....	13
3. INTERFACE REFERENCE GUIDE.....	14
3.1. WCT EXECUTION SERVLET	14
3.2. WCT CONFIGURATION SERVLET	14
4. TROUBLESHOOTING Q&A	16
5. CONTACT INFORMATION AND CREDITS.....	17
6. THE EDG LICENSE AGREEMENT	18

COPYRIGHT NOTICE

Copyright (c) 2005 by **Academic Computer Centre CYFRONET AGH**. All rights reserved.

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

The WCT product in its current version makes use of external libraries in its operations. Namely it depends on set of java libraries:

1. the **Axis** libraries provided by The Apache Software Foundation
2. the **log4j** library provided by The Apache Software Foundation
3. the **wsdl4j** library provided by The Apache Software Foundation
4. the **JDOM** xml parsing library provided by Jason Hunter and Brett McLaughlin
5. the **Commons** java class libraries provided by The Apache Software Foundation
6. the **StAX** streaming API for XML library provided by Codehaus
7. the **XFire** SOAP framework provided by Codehaus
8. the **Concurrent Utilities for Java** library provided by Doug Lea
9. the **JAF** framework provided by Sun Microsystems
10. the **JavaMail** library provided by Sun Microsystems
11. the **ServletAPI** library provided by Sun Microsystems
12. the **gomclient** library released with K-WfGrid license by ACC CYFRONET AGH
13. the **gworkflowdl** library released with K-WfGrid license by Fraunhofer FIRST
14. the **ICU** library provided by IBM
15. the **Jena Semantic Web Framework** provided by Jena Community
16. the **JUnit** framework provided by Object Mentor

All the above software is provided for use free of charge on the basis of open source licenses, either the Apache license (libraries 1 to 7), Sun Microsystems license (libraries 8 to 11), X license (library 14), Jena license (library 15), Common Public License or the K-WfGrid license (libraries 12 to 13).

Axis, log4j, wsdl4j is a registered trademark of The Apache Software Foundation. All rights reserved.

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

1. INTRODUCTION

The Workflow Composition Tool (WCT) is designed to automatically compose abstract workflows of Grid applications. The current version (1.3) which is described by this documentation forms the third release of this product. The tool is meant to be used in conjunction with other results provided by the K-WfGrid Project, especially the ontological service registry GOM (Grid Organization Memory – Work Package 4) and the Grid workflow language (GWorkflowDL) devised by Work Package 2.

The functionality of this tool enables the transformation of initial scientific workflows (described with use of GWorkflowDL language) into so-called abstract workflows. The tool uses rich semantic description of abstract operations registered in the environment in order to ensure validity of the proposed solutions. The integration of the tool with the K-WfGrid ontological standard helps the user to properly define request and to better understand the produced results. Although the WCT is meant to be used as a part of workflow execution environment being built in Work Package 2, it may also be used from command line. The detailed description of the mode of use is provided inside this manual. On the other hand, the accompanying developer's manual helps future designers with using this tool in their own software.

1.1. ABBREVIATIONS AND ACRONYMS

CTM – Coordinated Traffic Management

ERP – Enterprise Resource Planning

FFSC – Flood Forecasting Simulation Cascade

GOM – Grid Organization Memory

GWES – Grid Workflow Execution Service

GWorkflowDL – Grid Workflow Description Language

GWUI – Grid Workflow User Interface

K-WfGrid - The Knowledge-based Workflow System for Grid Applications

OWL-S – Web Ontology Language Services

VO – Virtual Organization

WCT – Workflow Composition Tool

XML – Extensible Markup Language

1.2. REFERENCES AND SOURCE CODE

The WCT tool's source code and JavaDoc code documentation may be found following this web link:

<http://www.cyf-kr.edu.pl/~ymgubala/wct/docs/>

The current state of WCT development in form of the source code tree may be accessed through the CVS server:

<http://cvs.ui.sav.sk/cgi-bin/cvsweb.cgi/kwfgrid/wct/>

2. PRODUCT USAGE

2.1. RUNNING THE PRODUCT

2.1.1. Operating Requirements

2.1.1.1. Local hardware requirements

The hardware requirements of the WCT tool are not very high as any computer hardware able to run Java 5 software is sufficient. Also the Internet connection is needed in order to make WCT work with external knowledge base. As the tool is quite communication dependant, the optimal configuration will have a low-latency Internet connection.

2.1.1.2. Local software requirements

The tool needs to be executed in Java 5 Runtime Environment which may be obtained free of charge from SUN's java website (<http://java.sun.com/j2se/1.5.0/>).

A list of needed compiled java libraries (in form of *jar* files):

- activation-1.0.2.jar (from <http://java.sun.com/products/javabeans/jaf/downloads/index.html>)
- axis-1.3.jar (from http://sunsite.icm.edu.pl/pub/www/apache/dist/ws/axis/1_3/)
- axis-jaxrpc-1.3.jar (from http://sunsite.icm.edu.pl/pub/www/apache/dist/ws/axis/1_3/)
- axis-saaj-1.3.jar (from http://sunsite.icm.edu.pl/pub/www/apache/dist/ws/axis/1_3/)
- commons-codec-1.3.jar (from <http://archive.apache.org/dist/jakarta/commons/codec>)
- commons-discovery-0.2.jar (from <http://archive.apache.org/dist/jakarta/commons/discovery/>)
- commons-httpclient-3.0.jar (from <http://archive.apache.org/dist/jakarta/commons/httpclient/>)
- commons-logging-1.0.3.jar (from <http://archive.apache.org/dist/jakarta/commons/logging/>)
- concurrent-1.3.4.jar (from <http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/current/>)
- mail-1.3.3_01.jar (from http://java.sun.com/products/javamail/javamail-1_3_3.html)
- servletapi-2.3.jar (from <http://java.sun.com/products/servlet/download.html>)
- stax-1.1.1-dev.jar (from <http://dist.codehaus.org/stax/jars/>)
- stax-api-1.0.jar (from <http://dist.codehaus.org/stax/jars/>)
- gom-client-api-1.0.jar (from <http://staff.science.uva.nl/~gubala/kwfgrid/jars/>)
- gomclient-20060510.jar (from <http://staff.science.uva.nl/~gubala/kwfgrid/jars/>)
- gworkflowdl-1.0.3.jar (from <http://www.gridworkflow.org/kwfgrid/jars/>)
- icu4j-2.6.1.jar (from <ftp://ftp.software.ibm.com/software/globalization/icu/2.6.1>)
- jdom-1.0.jar (from <http://jdom.org/downloads/index.html>)
- jena-2.3.jar (from <http://jena.sourceforge.net/downloads.html/>)
- junit-3.8.1.jar (from <http://junit.org/index.htm>)
- kowariclient-1.0.jar (from <http://staff.science.uva.nl/~gubala/kwfgrid/jars/>)
- log4j-1.2.8.jar (from <http://archive.apache.org/dist/jakarta/log4j/>)
- wsdl4j-1.4.jar (from <http://prdownloads.sourceforge.net/wsdl4j/>)
- xercesImpl-2.7.1.jar (from <http://archive.apache.org/dist/xml/xerces-j/>)
- xfire-all-1.0.jar (from <http://dist.codehaus.org/org.codehaus.xfire/jars/>)

2.1.1.3. Grid infrastructure requirements

The main requirement of WCT to ensure its proper functioning is the connection to an instance of Grid Organization Memory (GOM) registry. The registry has to be operational and it needs to publish its WebService lookup interface externally. Without support of that software the WCT is not able to effectively process any requests. No other specific hardware or software Grid infrastructure is needed.

For information where to setup the WCT-GOM connection check the Step-by-Step Setup section of this documentation.

2.1.2. Step-by-Step User Setup

Step 1: download WCT tool jar from <http://www.cyf-kr.edu.pl/~ymgubala/wct/> (also files `wct.props` and `log4j.properties` will be important – download them as well)

Step 2: make sure you have a proper installation of Java 5 Runtime Environment on your local site (simple check for java executable would do). If not, download it and install properly.

Step 3: get all the software dependencies of the tool (in form of a list of jar files). Please check *Local software requirements* section for locations of these files.

Step 4: identify the web endpoint URL where an instance of GOM registry is available and supply it to the WCT properties file – set the `REGISTRY_ENDPOINT_ADDRESS` property value in `wct.props` file for the one identifying the GOM web interface.

2.2. BASIC OPERATION

The main interface of the tool is able to transform the initial user requirement into an instance of abstract Grid workflow. In order to use the interface from the command line one has to execute a class of Composer inside the Java Runtime. Below there is an example of composer execution:

```
java net.kwfgrid.wct.Composer init-workflow.xml VO-names
```

Before issuing that command please make sure all the jar libraries mentioned in the *Local Software Requirements* section are accessible through the Java classpath (either using the `$CLASSPATH` environment variable or with the `-cp` option). Also it is important to include both `log4j.properties` and `wct.props` files in the directory accessible from the Java classpath list. For convenience a `cpwct` file contained in the distribution package is provided with all necessary java classpath entries. To use it the invocation of the tool should look as follows (for Linux machines):

```
java -cp `cat wct/cpwct`:wct/wct-x.x.x.jar \  
net.kwfgrid.wct.Composer init-workflow.xml VO-names
```

The first argument of the execution is an XML file that contains the initial workflow description. The only workflow description standard acceptable for the WCT tool at the moment is the GWorkflowDL notation. For a thorough information on that standard please check its development webpage (<http://www.gridworkflow.org/kwfgrid/gworkflowdl/docs/>). Nevertheless a short description on how to form a simple initial workflow is presented in this document as well (see next subsections).

The last argument of the Composer's execution is the list of names of Virtual Organizations. The list is treated by the tool as sources of possible operations that may be included into the resultant workflow. If a certain VO should make its resources available to a certain application, the user has to include this VO's name in the VO-names list (the particular names are separated with the semicolon, e.g. "CTM;ERP"). As the names of the organizations are identical to the names of sections inside the GOM repository, it is natural to ask your contact person to the GOM installation for the exact names of the VOs that one should use for this execution argument.

Having provided a set of examples of initial workflows, one may use them to run the Composer and to check whether the installation and the setup is proper. The examples may be downloaded from <http://www.cyf-kr.edu.pl/~ymgubala/wct/examples/> website.

```
java net.kwfgrid.wct.Composer ctm-red-wf.xml CTM
```

This shows the execution of the initial workflow from the Coordinated Traffic Management (CTM) K-WfGrid application – please note the proper name of the VO (CTM).

2.2.1. Short introduction to initial workflows in GWorkflowDL

This short tutorial explains how to prepare a new, simple initial workflow that may be supplied to the WCT instance. Please have a look onto the example below:

```
<workflow owl="">
  <place ID="begin">
    <token>
      <xsd:boolean>true</xsd:boolean>
    </token>
  </place>
  <place ID="result">
    <tokenClass type="effect" owl="http://gom.kwfgrid.net/gom/ontology/
      DomainServiceOntology/CTM#TrafficCongestionReduced"/>
  </place>
  <transition ID="GridJob">
    <inputPlace placeID="begin"/>
    <outputPlace placeID="result"/>
    <operation/>
  </transition>
</workflow>
```

This example shows a simple workflow based on PetriNets formalism. It contains two *places* and a single *transition*. The transition joins the two places the way that the “begin” place points towards the transition, which on the other hand points towards the “result” place. The most of the parts of this file are reusable for definition of a new initial workflow – the most important is the description of “result” place. It is the place where a user defines what result should be produced by the application workflow that should be composed by the WCT Composer.

There could be two types of results: *effects* and *data*. The first case is represented by the example (what is indicated by “effect” type of the tokenClass element of the place called “result” - the other case is identified by the “data” value of this argument). In both cases the user needs to provide the identifier of the effect/data that one wants to achieve through the workflow execution. This Id is

provided as the value of the owl attribute. This identifier has to be defined by the GOM ontology registry that the installation is using. As eligible candidate one may assume any individual of the ontological class *SimpleEffect* defined by K-WfGrid ontology structure (see the GOM manual for more details on how to search for such individuals). In the case that one wishes to obtain certain data, (s)he should use Id of an individual of the ontological class *DataObjectOutput* defined by the mentioned ontology structure (or, a simpler class of *Output* defined by the OWL-S Process standard).

2.2.2. Simple workflow patterns in WCT

In order to exemplify the operations of workflow composition algorithm that is hidden inside the tool, we supply below a short discussion on how (given certain circumstances) the composer would solve a particular dependency situation.

Let us assume we deal with a single data dependency (it is irrelevant for this discussion whether it is a data or an effect dependency) called *Dep-A* (see Figure 1). According to the current rules of GWorkflowDL language, this is represented as a simple PetriNet with one unknown (but foreseen) operation (the red transition rectangle) and two places. The upper black place is called a *control* place as it is there just for PetriNet well-formedness reasons, but the other, yellow one is a data place that is intended to hold a piece of data during the workflow execution.

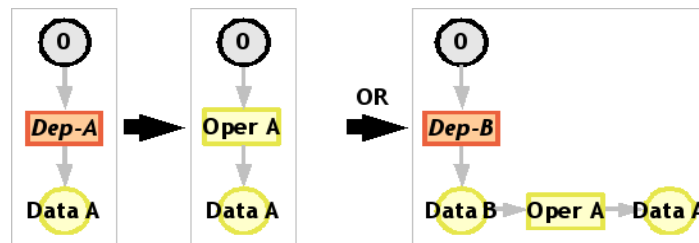


Figure 1: Possible simple solutions to a dependency

Please assume there is an abstract service registered inside the knowledge base that provides an operation which is able to produce the needed data *Data A*. In that case we have a solution but the future structure of the workflow after the refinement step depends on the characteristics of the found operation *Oper A*. If the operation does not require any input information to work, the solution would be the simplest one (pictured in the middle of Figure 1). Another example in the picture shows the situation if the newly introduced operation needs a certain *Data B* input. In that case a new data dependency has to be created (called *Dep-B*) and solved later.

In the Figure 2 one may see another possible solution, this time involving more complex workflow patterns. In the left-hand case the WCT found two equally good provider operations. As from the functional point of view these two are indifferent, both solutions are included into the workflow in form of XOR-split/merge pattern. It is a responsibility of further, run-time refinement tools to choose which operation to during the execution phase. On the other hand the right-hand case in the Figure 2 shows a situation that emerges when a solution operation needs more then one input. In this case every needed input spawns a new data or effect dependency and both these parallel paths are joined together in a single AND-split pattern (the horizontal bar in the picture).

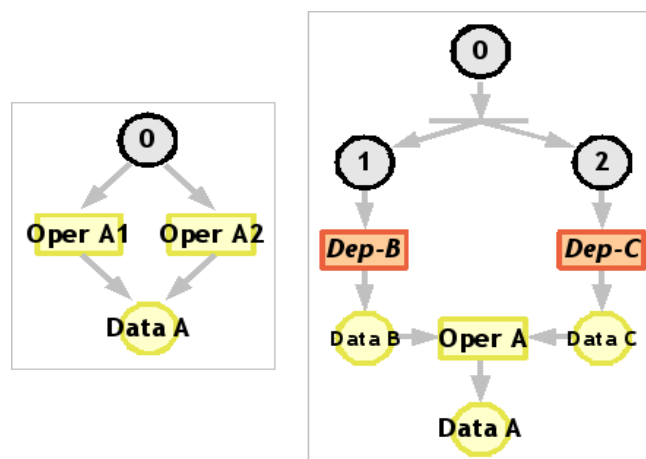


Figure 2: Possible complex solutions to a dependency

2.2.3. WCT Optimization algorithms

During setting up the WCT tool one may control the usage of optimization algorithms by enabling or disabling them. Implementation of such algorithms was imposed by observing often workflow patterns emerge during workflow composition. The first one prevents the workflow to spread by duplicating workflow sections, as a result of similar dependencies in one AND branch, while the second one minimizes, if certain conditions are met, number of dependencies in the final workflow. Following is a description of AND-branch optimization and dependency reduction algorithms.

2.2.3.1. AND-branch optimization algorithm

The left part of Figure 3 shows an example of an AND-split workflow pattern with three branches composing it. Such a construction may represent parallel application processing with three execution paths synchronized at the and by operation D (*Op D*). This particular example has three dependencies and it can be noticed that two of them (those marked with green areas) require the same data, denoted by A. When resolved, the workflow would contain two identical branches as a result of processing the mentioned dependencies. This, during the application execution, would cause unnecessary resource load and if the duplication of branches was large, compared to the size of workflow, application's efficiency would drop dramatically. To prevent such cases AND-branch optimization algorithm has been introduced. After each step of WCT the algorithm is applied to the current workflow and all matching AND-splits are reduced according to the manner presented in Figure 3. For each AND-split pattern similar dependencies are found and transferred into one dependency. Data required by operations down the execution path are copied from the newly created dependency by using the edge expression copying mechanism. This results in marking the edges of the workflow between source and destination places holding the required data, just as in the right part of Figure 3.

The structure of the processed workflow is changed within limits of transformed AND-splits. Outside the workflow stays unchanged. Unprocessed dependencies (in this example the third one not marked with green area) as well as paths after new dependencies keep their parallel characteristic and so the application execution manager can still use it as benefit.

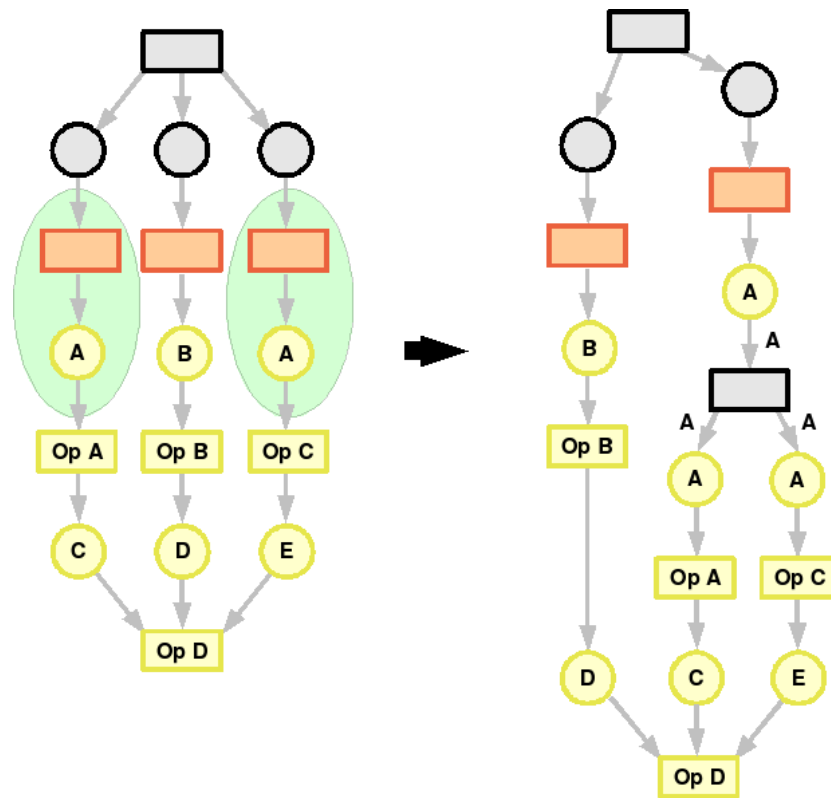


Figure 3: AND-branch optimization

2.2.3.2. Dependency reduction algorithm

In Figure 4 an example of dependency reduction is depicted as a two-step process. The algorithm which is used to transform the workflow is applied at the end of composition process. In that phase a fixed number of dependencies is present in the workflow, which is a result of too few WCT tool steps or not finding appropriate operations in the application's domain. To successfully execute such application some extra data need to be provided by the end user for each unresolved dependency. However, it was observed that certain dependencies occur in several places in one workflow, as in Figure 4. In the left-most workflow two dependencies which require data referred here as **B** are present and preceded by a series of black places and transitions towards the beginning of the workflow. It would be convenient to move the dependency to the beginning of the workflow and use the black elements to copy the provided data and reducing the number of transitions. This is done by *Dependency reduction algorithm*, which roughly is described below.

The algorithm starts with grouping all dependencies present in the workflow according to the data which they produce. Among the groups the ones with the largest sizes are chosen and processed in the first place. This assures maximal dependency reduction as a group has the biggest chances of being reduced at the beginning when there still is a big number of black places. For each dependency in one group the path towards the beginning of the workflow is analyzed. If begin place or a non-AND black transition is reached along the path the dependency is moved up and the old one is replaced by a single data place, as in the middle workflow in Figure 4. The edges along the path are marked and black places are transferred to data places. For any other dependency in the group the process is repeated. In the presented example analyzing the path showed the existence of a place holding the required data. In such case dependency is removed and edge expression copying mechanism is used to deliver the required data to the operation B (*Op B*, see right-most workflow in Figure 4). It is worth mentioning

that paths with black elements used for one group reduction cannot be used for a different one, as data places can hold only one type of data token.

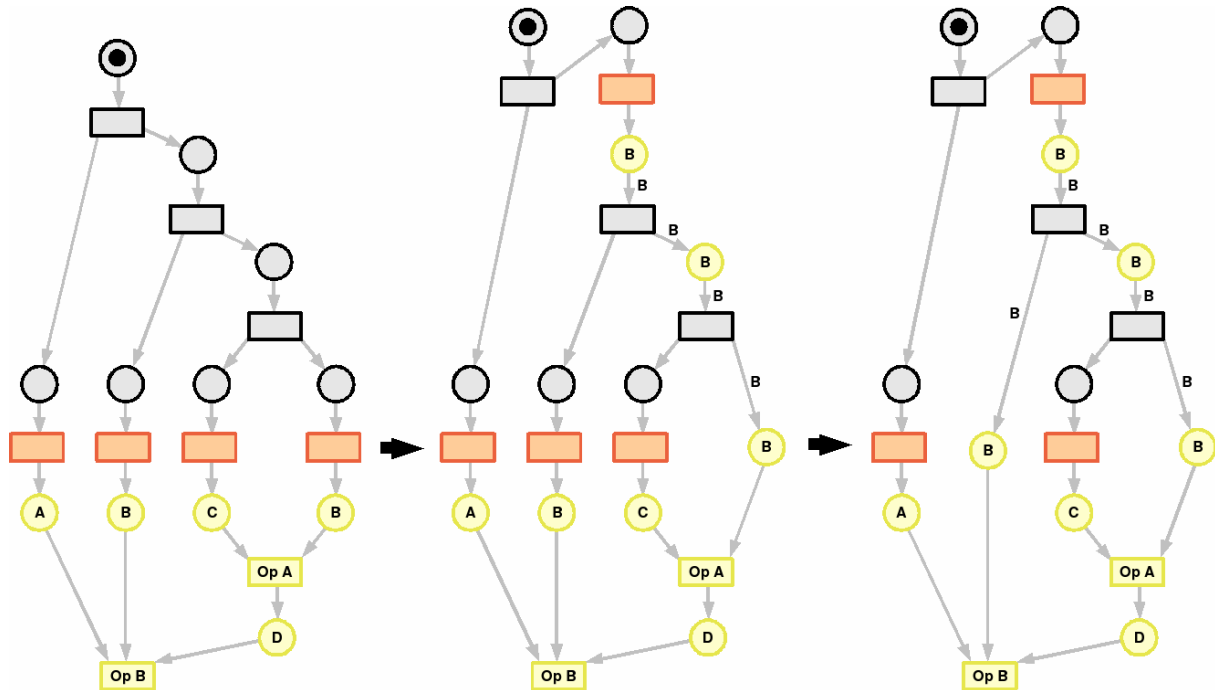


Figure 4: Dependency reduction example

2.3. ADVANCED FEATURES

The WCT tool besides the command line use described in the *Basic Operation* section can also be invoked through a web service. Below is a description of enabling the tool as a web service and a simple example of its use.

2.3.1. Deploying WCT tool as a Web Service

The WCT software can be published as a web service endpoint and be enhanced with its interoperability advantages. The framework used as the SOAP protocol implementation is *Axis* and all necessary libraries are distributed with the WCT distribution as a ready-to-use war file (*wct.war*). Therefore no additional *Axis* installation is needed. The described installation is specific for the Tomcat web server and it was tested with the 5.5.17 version of Tomcat distribution.

To deploy the war file simply copy it to the web application directory of the Tomcat server (usually *\$CATALINA_HOME/webapps*). If the Tomcat is configured to automatically load war contents the application should be available through address <http://localhost:8080/wct> of a standard Tomcat installation (if not, please change the address according to you local Tomcat configuration). The browser should display two links which navigate to *Execution* and *Configuration* servlets. To activate the web service endpoint *Axis* framework needs to be notified by using the admin client. This can be done using the following command:

```
java -classpath $AXIS_CLASSPATH \
  org.apache.axis.client.AdminClient \
  -s/wct/services/AxisServlet deploy-wct.wsdd
```

The `$AXIS_CLASSPATH` points to all necessary jar files needed by the framework. They can be obtained from the Axis project page (<http://ws.apache.org/axis/java/releases.html>) or extracted from the `wct.jar` file. The `deploy-wct.wsdd` is the deployment descriptor file and is distributed together with the WCT package.

To check if the WCT web service is operational use the following address:

<http://localhost:8080/wct/services/IWfConstructionService?wsdl>

This should result in a wsdl xml file of the WCT service.

2.3.2. Using WCT through a web service interface

To use the WCT web service using the Axis framework a proper client code has to be executed. Such client is implemented as one of the WCT's classes and can be used by invoking the following command:

```
java -classpath wct-x.x.x.jar:$AXIS_CLASSPATH \  
  net.kwfgrid.wct.client.WctClient workflow-file.xml \  
  http://localhost:8080/wct/services/IWfConstructionService \  
  http://localhost:8080/wct/services/IWfConstructionService \  
  composeAbstractWorkflow CTM
```

There are 4 arguments which need to be provided for successful client execution. First is the workflow filename, second is the web service endpoint address, then web service namespace needs to be specified and operation name as the third and fourth argument. The fifth optional argument is a semicolon separated list of virtual organization names.

2.4. KNOWN PROBLEMS

The current version (1.3) of the WCT software is its third release. Most of the functionality is considered stable. However, lately new beautifying and reducing algorithms have been introduced and still are in testing phase. For certain workflows they may still produce unexpected results.

3. INTERFACE REFERENCE GUIDE

In the third release of WCT application a prototype of a web interface was added. The functionality of the interface includes workflow composition and WCT tool configuration routines. It has been implemented as two separate servlets which usage is described below.

3.1. WCT EXECUTION SERVLET

Standard WCT's web configuration exposes execution servlet under the address <http://host:port/wct/execute> where *host* is the name of the machine running the servlet container and *port* is the port number through which the container is available (depending on the server configuration). The layout of the servlet's form is presented in Figure 5. Two input fields are present to provide data required for execution. The large text area is used to put into the initial workflow and beneath a text field can be optionally filled in with semicolon separated list of virtual organisation names.

After the execution button at the bottom of the servlet's page is pressed data is transferred and workflow composition is processed, which may take a significant amount of time. As a result an xml document is returned with the final workflow.

```
<?xml version="1.0"
encoding="ISO-8859-1"?>
<workflow
xmlns="http://www.gridworkflow.org/gwork
xmlns:xsi="http://www.w3.org/2001/XMLSchema
xmlns:xsd="http://www.w3.org/2001/XMLSchema
xsi:schemaLocation="http://www.gridworkf
http://www.gridworkflow.org/kwfgrid/src/
ID="No_ID">
  <property
name="domain">CTM</property>
  <place ID="begin">
    <token>
      <control
xsi:type="xsd:boolean">true</control>
```

Figure 5: WCT execution servlet page

3.2. WCT CONFIGURATION SERVLET

Property name	Old value	New value
PERFORMANCE_DATA_FILE	performance.out	

Figure 6: WCT configuration servlet page

To tune the WCT tool number of properties need to be set. To make the process convenient a configuration servlet has been implemented and in the standard WCT configuration exposed with the address <http://host:port/wct/configure> where *host* and *port* are name and port of the server with the WCT application running. The page with the configuration form, depending on the operating system

and browser may look more or less as in Figure 6. Properties can be chosen through the left-most drop-down list. After choosing the name of the property and submitting the form by clicking the *Update* button the value of the property is loaded and displayed in the *Old value* column. To change it to a new value the field under *New value* table heading needs to be filled in with a non-empty string value and again *Update* has to be activated. The newly inserted value should then be displayed in the *Old value* field and the new value field is empty and ready for a next property change.

4. TROUBLESHOOTING Q&A

Q: The tool execution finishes with an exception reporting lack of connection with the registry contact.

A: The WCT is not able to work without an operational connection to an instance of GOM service registry. In order to set up one, please refer to Grid Organization Memory user manual (supplied by the K-WfGrid Consortium as well). If you know there is a working instance of this registry already supplied on the Web, please look onto *Step-by-step user setup* section in order to learn how to inform the WCT of this fact.

Q: The tool complains that it cannot find definitions of some classes (*DefNotFoundError*).

A: Some of the needed libraries are not loaded properly or are simply not present. Please make sure you have all the jar files (mentioned in Local software requirements section) and that you load them properly into the Java Virtual Machine. The libraries would preferably be in the listed versions, although there is a fair chance that similar (newer) versions will be sufficient as well.

Q: The tool execution result in a JDOM library exception that it cannot parse the file properly.

A: There are probably problems with the initial workflow description file. Please, make sure it is formed in the way described by the Basic operation section. If you think you defined the workflow right, please check GWorkflowDL website (www.gridworkflow.org/kwfguid/gworkflowdl/docs/) to obtain the current schema of the workflow notation.

5. CONTACT INFORMATION AND CREDITS

The authors of the Workflow Composition Tool:

- Tomasz Gubała (gubala@science.uva.nl)
- Daniel Hareźlak (D.Harezlak@cyfronet.pl)

We ask to send all the remarks and bug reports to any address from the above list.

The authors also would like to express their gratitude for all the contributors:

- Maciej Malawski (from AGH University of Science and Technology, Kraków, Poland)
- Bartosz Kryza (from ACC Cyfronet AGH, Kraków, Poland)
- Marian Bubak (from AGH University of Science and Technology, Kraków, Poland)
- Andreas Hoheisel (from Fraunhofer FIRST Institute, Berlin, Germany)

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.