The Java CoG Kit User Manual
Version 4.0-pre-alpha

MCS Technical Memorandum ANL/MCS-TM-259

Revisions March 14, 2003, July 18, 2003, Nov 28, 2003, Jan 14, 2004

*The Java CoG Kit Team
Argonne National Laboratory
Mathematics and Computer Science Division
9700 S. Cass Ave
Argonne, IL 60439

* Coresponding Editor
(630) 252 0472
gregor@mcs.anl.gov

Location of Manual:

http://www.globus.org/cog/manual-cog2.pdf

Be kind to your environment and
do not print
this frequently changing manual.

March 9, 2004

Contents

1 Preface **7**

1.1 Participation Opportunities 7

1.2 Grids 8

1.3 Intended Audience 8

1.4 Resources 9

1.4.1 Project Website 9

1.4.2 Bug Reporting 9

1.4.3 Mailing Lists 10

1.4.4 Sourcecode Repository 11

1.5 About the Manual 11

1.5.1 Conventions 11

1.6 Manual Maintainer 12

1.7 Contributors 13

1.8 Administrative Contact 13

1.9 Acknowledgments 13

2 License **14**

2.1 Project Registration 14

2.2 Globus Toolkit 16

2.2.1 Globus Toolkit Public License (GTPL) 16

2.2.2 Globus Toolkit Contributor Liceense 18

2.3 Java CoG Kit 20

2.3.1 Java CoG Kit Public License (JCoGPL) 20

2.3.2 Java CoG Kit Contributor Liceense 22

2.4 Other Licences 24

2.4.1 jglobus 24

2.4.2 ogce 24

2.4.3 Others 24

2.5 GNU Public Licence 24

3 Introduction **25**
3.1 Overview . . . 25
3.2 History . . . 25
3.2.1 Metacomputing . . . 25
3.2.2 CoG Kits . . . 25

4 Installation **26**
4.1 Download . . . 26
4.2 CVS Release Tags . . . 26
4.2.1 Prerequisites . . . 26
4.3 Downloading the Java CoG Kit . . . 26
4.4 Compiling the Java CoG Kit . . . 27
4.5 Compiling the complete distribution . . . 27
4.6 Compiling individual modules . . . 27
4.7 Using the Java CoG Kit . . . 27
4.8 Setup . . . 28

5 Contributing **29**
5.1 Creating a module . . . 29
5.1.1 Build files . . . 29
5.1.2 Libraries . . . 29
5.1.3 Source . . . 30
5.1.4 Using PMD . . . 30
5.1.5 Documenting the modules . . . 30
5.1.6 Maintaining a module . . . 30
5.1.7 Launchers . . . 30
5.1.8 Webstart . . . 30
5.2 Coding Guidelines for the Java CoG Kit . . . 30
5.2.1 Imports . . . 30
5.2.2 Indentation . . . 30
5.2.3 Brackets . . . 30
5.2.4 Variables . . . 30
5.2.5 Instance Variables . . . 31
5.3 One-Liners . . . 31
5.3.1 Logging . . . 31

5.3.2 Testing 31
5.3.3 Internationalization 31
5.3.4 Library Reuse 31
5.3.5 Exceptions 31

6 Modules **32**
6.1 util 32
6.2 certrequest 32

7 Core **33**
7.1 Introduction 33
7.2 Installation 33
7.2.1 Download 33
7.2.2 Compile 34
7.2.3 Configuration 34
7.2.4 Examples 34
7.3 Design 35
7.3.1 ExecutableObject 36
7.3.2 Task 36
7.3.3 Specification 37
7.3.4 TaskGraph 40
7.3.5 Status 41
7.3.6 Handlers 42
7.4 Programmer's Guide 44
7.4.1 Executing a remote job execution task 44
7.4.2 Executing a third party file transfer task 48
7.4.3 Executing a simple TaskGraph (DAG) 50
7.4.4 Executing a hierarchical TaskGraph 52
7.4.5 Writing a custom TaskHandler 53
7.5 jglobus 54

8 Karajan **55**
8.1 Installation 55
8.1.1 Obtaining the Source Code 55
8.1.2 Compiling Karajan 55
8.2 Using Karajan 56

8.2.1 Command Line Interface 56
8.2.2 Graphical Interface 56
8.3 Language Specification 58
8.3.1 Concepts 58
8.3.2 Parallelism 62
8.3.3 Iterators 63
8.3.4 Templates 65
8.3.5 Grid-related Elements 65
8.3.6 Explicit Error Handling 68
8.3.7 Miscellaneous Elements 68
8.3.8 Functions 69
8.4 Supported Handlers 69
8.5 Include Search Path 71
8.5.1 System Defaults 71
8.6 Architecture 72
8.6.1 The Loading Process 72
8.6.2 The execution model 72
8.6.3 Task Scheduling 74
8.7 Checkpointing 75
8.7.1 Checkpoint Creation 75
8.7.2 Restoring from a Checkpoint 76
8.8 Quick Element Reference 77
8.8.1 allocateHost 77
8.8.2 checkpoint 77
8.8.3 default 77
8.8.4 echo 77
8.8.5 elementDef 77
8.8.6 executeJava 78
8.8.7 for 78
8.8.8 foreach 78
8.8.9 function:contains 78
8.8.10 function:numberFormat 78
8.8.11 function:readFile 79
8.8.12 function:UID 79
8.8.13 generateError 79

8.8.14 gridExecute 79
8.8.15 gridTransfer 79
8.8.16 ignoreErrors 80
8.8.17 include 80
8.8.18 nonCheckpointable 80
8.8.19 javaBean 80
8.8.20 javaBean>invokeVoid 80
8.8.21 javaBean>setProperty 80
8.8.22 parallel 81
8.8.23 project 81
8.8.24 restartOnError 81
8.8.25 scheduler 81
8.8.26 scheduler>grid 81
8.8.27 scheduler>grid>host 81
8.8.28 scheduler>grid>host>service 81
8.8.29 scheduler>taskHandler 82
8.8.30 scheduler>taskHandler>securityContext 82
8.8.31 scheduler>taskHandler>securityContext>property 82
8.8.32 sequential 82
8.8.33 setvar 82
8.8.34 template 82
8.8.35 templateDef 83
8.8.36 wait 83

9 Graph Editor **84**
9.1 Configuring 84
9.2 Running 84
9.3 Using The Graph Editor 84
9.3.1 The Swing Target 85
9.3.2 The HTML Target 86
9.3.3 The PostScript Target 87
9.3.4 The Remote Target 88
9.4 Graph file format 88
9.5 API 89
9.6 Scalability 91

10 Portlet **92**
10.0.1 Deployment 92
10.0.2 Directory structure of portlet module 93
10.0.3 Prerequisite - Jetspeed 93
10.0.4 Prerequisite - MyProxy 94
10.0.5 Configuration 94
10.0.6 Installation 95

11 Java CoG Kit QoS Module **96**
11.0.7 QGS Installation Prerequists 96
11.0.8 QGS Compilation and service deployment 96
11.0.9 Examples 97
11.1 Resources 98
11.2 Common 98
11.3 All 98

12 Command Tools **99**

13 grid-cert-request **100**

1 Preface

This manual contains a number of high-level modules of the Java CoG Kit that are not distributed as part of the Globus Toolkit version 2, 3, or 4. We believe these components are valuable add-on components to any Grid Toolkit. The popularity of the Java CoG Kit has lead to the fact that it is now distributed in part with the Globus Toolkit. Often users of the Globus Toolkit, do not know that they use components contributed by the Java CoG Kit.

We hope that you will find the components described in this manual help you making use of the Grid more easily.

The Java CoG Kit is a very open project, and invites participation by others. Thus, we have started to involve the community more strongly into the development of the Java CoG Kit. If you have components that you like to contribute to the Java CoG Kit, please notify us.

If you like to participate in the development of the Java CoG Kit, I recommend that you contact us through a simple e-mail as described in Section 1.1.

1.1 Participation Opportunities

To participate, please send a mail with your intend and abilities to

Gregor@mcs.anl.gov : .

Please, follow the simple subject mail syntax rule

“APPLICATION CV: <Firstname><Lastname>”,

where CV is an abbreviation for community volunteer. All mail not following this rule will be caught by a spam filter and automatically deleted.

We have a variety of open projects that could provide ideal opportunities to get engaged in furthering the development of Grid computing. Some of these projects could also be given for credit as independent studies, or lead to a Masters Thesis project. If you decide to integrate them in your curriculum it is best to develop an agreement between the Java CoG Kit project, your advisor, and yourself. We conducted such activities with volunteers from Canada, UK, Switzerland, and several local and remote students and professionals in the US.

Doe to the nature of volunteering, these applications are usually less formal than real applications, but we must know your affiliation, your address and citizenship. To volunteer, you ought to be committed. It is of no help to us if you volunteer one week and than you drop the project in the next week.

If you apply for community volunteer positions make sure you provide us with evidence that you can conduct the project you or we suggest.

Community volunteer projects are a good start for a paid internship or job opportunities. Other paid opportunities for undergraduate and graduate appointments are updated regularly on the Web page. Often it is a good idea to have your advisor directly talk to us and recommend you over the phone, or in one of the many meetings we participate in. Paid assistantships are in general restricted to US citizens and permanent residents.

All contributors have to submit a contributor license.

1.2 Grids

Grids are an important development in the discipline of computer science and engineering. Rapid progress is being made on several levels, including the definition of the terminology, the design of an architecture and framework, the application in the scientific problem solving process, and the creation of physical instantiations of Grids on a production level.

A small overview about the Grid can be found in a draft paper entitled *Gestalt of the Grid* [1]

Article : `http://www.mcs.anl.gov/~gregor/bib/papers/vonLaszewski--gestalt.pdf`

This article provides an overview of important influences, developments, and technologies that are shaping state-of-the-art Grid computing.

What motivates the Grid approach?
What is a Grid?
What is the architecture of a Grid?
Which Grid research activities are performed?
How do researchers use a Grid?
What will the future bring?

A slightly differnt focus on middleware is presented in a paper entitled "Grid Middleware" [2]

Article : `http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-gridmiddleware.pdf`

Other CoG Kit related papers can be found at

References von Laszewski : `http://www.mcs.anl.gov/~gregor/bib/`

[1]

1.3 Intended Audience

This manual is intended for the intermediate Grid programmer that would like to access the Globus Toolkit functionality through Java. We assume that the reader of this manual is familiar with Java. If not, general information about Java is available through the Web site at SUN Microsystems or at IBM:

SUN : `http://java.sun.com/`

IBM : `http://www.ibm.com/java/`

[1] the bib file needs to be updated. Also there is a collection at www.cogkits.org

In general, this manual serves as a basic introduction to a subset of functionality provided by the Java CoG Kit. This manual does not explain every package, class, and method. This manual is intended to show you that the Java CoG Kit provides an effective way of accessing the Grid through Java.

Developers are encouraged to inspect the JavaDoc documentation. [2]

We further expect that you are familiar with the Globus Toolkit and have access to a Globus Toolkit installation. If you do not, the Globus web page provides information about the details and how to install it.

Globus Toolkit : `http://www.globus.org`

The Globus Toolkit development us undergoing some significant changes. If you currently use Globus Toolkit 2.4.x, we do recommend to evaluate a switch to version 3.2 carefully. This is in anticipation the Globus Toolkit version 3.2 will be replaced with Toolkit version 4.x during the year 2004. The Java CoG Kit provides so far an abstraction that protects the application user from the differences between these versions.

In case you develop with the Java CoG Kit APIs a switch between versions of the Globus Toolkit is simplified.

1.4 Resources

We support our efforts through a web site on which you find a bug tracking system, Mailing lists, and the code repository.

1.4.1 Project Website

Online information about the Java CoG Kit can be found on its home page.

Home page : `http://www.globus.org/cog/java/`

Here you can find links to the manual, the code, and some basic information about the project. Besides this page we also maintain a project-related Web page that reports on the Java and Python Commodity Grid Kits.

Project : `http://www.cogkits.org/`

1.4.2 Bug Reporting

We are using the Bugzilla system from mozilla.org to track bugs and requests for enhancements for the Java CoG Kit. Bugzilla provides you with an interface that guides you on submitting the bug. The link to the bug system is located at

CoG Kit Bugzilla : `http://www.globus.org/cog/contact/bugs/`

In case you like to report bugs for other components of the Globus Toolkit you can use the main link at

Globus Toolkit Bugzilla : `http://bugzilla.globus.org/globus/`

[2] we need to make sure that we have in the ant script a publication mechanism of the JavaDoc. We need to document how we update the web page, and manual. E.g. in the doc directory we say “make ; make publish

To use it you need to first create an account. To report a bug you need to be precise in your description and include operating system, JVM version, and other information that can be used to better identify or replicate the condition of your error. This also includes the version of Globus Toolkit services you use.

1.4.3 Mailing Lists

We have established a number of mailing lists to simplify the communication with the group of developers and users. Restrictions on the use of the mailing list are outlined below.

Policy

No Advertisements : We do not allow you to use the mailing lists in any form of advertisement for your products or services. In response to spam mail on this mailing list, we have disabled the ability to post messages to this list if you are not subscribed to it.

Subscription Required : If you send a message to the list and are not subscribed or you use an email address different from the one you subscribed with, your message will not be posted to the list, and you will not receive any notification that your message was *not* posted. Hence, if you send a message to the list and do not subsequently see your message on the list or in the list archive, verify that you are using an email address that is subscribed to the list, and then retry your posting.

Subscribed Lists : To verify that you are subscribed to the list, send an email message from the email account you subscribed from to `majordomo@globus.org` with the single word "which" in the body of the message. You will receive in response a message listing the lists to which your email address is subscribed. If this mailing list does not appear in the list you receive, you are probably subscribed to the list under a different address and you will not be able to post messages to the list using your current address.

Subscription Center

If you would like to be notified of CoG Kit release updates, visit our convenient subscription center at

Subscribe : `http://www.globus.org/cog/contact/`

Other Globus related mailing lists can be found on the Globus web page

Subscribe : `http://www.globus.org/about/subscriptions.html`

Note that you can use these web pages to unsubscribe from the lists. All mailing list are maintained with majordomo. However, we did have to disable the *who* function in order to protect the members from spam bots.

News

News about the Java CoG Kit is sent in irregular intervals (the frequency is monthly to every four month) by means of the following list:

CoG News : `cog-news@globus.org`

Sorted by Thread: : http://www-unix.globus.org/mail_archive/cog-news/threads.html

Sorted by Date : http://www-unix.globus.org/mail_archive/cog-news/maillist.html

Discussions and Community Developers

Discussions and general questions can be send to the high-volume e-mail list at

Java List : java@globus.org

Sorted by Thread : http://www-unix.globus.org/mail_archive/java/threads.html

Sorted by Date: : http://www-unix.globus.org/mail_archive/java/maillist.html

Note that this list may result in daily mails sent by the Java CoG Kit community. Please use the bug tracking system for reporting bugs. If you use the bug tracking system, your message has a higher chance of being answered. There is no guarantee that we answer a mail sent to the Java CoG Kit mailing lists.

1.4.4 Sourcecode Repository

We maintain all source code in a CVS repository that can be accessed anonymously. You can find more details about this in Section **??**.

1.5 About the Manual

This manual is constantly being improved and your input is highly appreciated. Please report suggestion, errors, changes, and new sections or chapters through our bugzilla system.

When you report bugs, please do not use page, line, or section numbers. Remember new sections may appear due to community contributions. Instead, please quote the section title, or make corrections by hand and FAX it to us. Even better, submit a corrected document, as you can check out the manual through our CVS archive.

1.5.1 Conventions

If you see a ?? or a ... in the text there is no reason to send us a report on it. It simply means that the section to which we refer has not yet been integrated in this manual. Comments that indicate issues that needs to be don, are included as footnotes. New text that has not yet been reviewed, may be in a different color. Regular text is written using the Times font. Code examples are highlighted in shaded blocks.

```
int a;
a = 1+ 2;
```

Interactive commands issued by a user in a shell are preceded with a > at the beginning of the line.

```
> ls
```

In case interactive commands exceed the 79 character limit, they are wrapped into the next line and are not proceeded by the > character. A backslash is included at the end of such lines to explicitly indicate that the command ins continued on the next line.

```
> echo "This is s very long text that is continued on the \
       next lines. The leading blanks in the next lines \
       are to be ignored"
> echo "This is a new command"
```

References to variables or other important text that is part of a program or shell script is written in `Courier`. To illustrate this on an example:

> Hence, a reference to the variable `int a` form our previous example uses also the `Courier` font.

Generic entities are wrapped between angle brackets. Each such entity is not to be taken literally. In general, such constructs are explained as they occur throughout the manual. The use of such entities is shown in the example below:

```
<machine-name>
```

Here, <machine-name> is to be replaced with an actual machine name:

```
> ping hot.mcs.anl.gov
```

Web links are proceeded by a meaningful name for the link. An example is

Java CoG Kit Website : `http://www.globus.org/cog`

Links to code source are proceeded by the repository tag. An example is

jglobus : `org/globus/gram/Gram.java`

1.6 Manual Maintainer

A number of people are currently maintaining the manual.

Part	Section	Name
Preface	1	Gregor von Laszewski
Introduction		Gregor von Laszewski
Licence		Gregor von Laszewski
Installation	4	Mike Hategan
Setup	4.8	Mike Hategan
Contributing	5	Mike Hategan
Modules		
jglobus		TBD
Util		TBD
Certrequest		TBD
Resources		TBD
Common		TBD
Grapheditor	9	Mike Hategan
Karajan	8	Mike Hategan
Core	7	Kaizar Amin
Portlet	10	Mike Hategan
QoS	11	Rashid Al-Ali
Command Tools		
Certrequest	13	Gregor von Laszewski

We invite you to contribute to the manual or the code (see 1.1).

1.7 Contributors

Gregor von Laszewski, Argonne National Laboratory, University of Chicago
Kaizar Amin, University of North Texas, ANL
Mike Hategan, University of Chicago, ANL
Shashank Shankar, Illinois Institute of Technology, ANL
Vladimir Silva, IBM
Jean-Claude Cote, High Performance Computing, National Research Council, Canada

If we have forgotten to include your name in the list of contributors please notify us.

1.8 Administrative Contact

The project is managed by Gregor von Laszewski. To contact him, please use the information below.

Gregor von Laszewski
Argonne National Laboratory
Mathematics and Computer Science Division
9700 South Cass Avenue
Argonne, IL 60439
Phone:(630) 252 0472
Fax: (630) 252 1997
gregor@mcs.anl.gov

1.9 Acknowledgments

This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-Eng-38. DARPA, DOE, and NSF support Globus Project research and development. This work would not have been possible without the help of Ian Foster and the Globus Project team.

2 License

The Java CoG Kit is distributed under two licenses. The parts that are included in the Globus Toolkit are distributed under the Globus Toolkit Public License (GTPL), which is listed in Section 2.2.1. The parts that are not distributed in the Globus Toolkit, are distributed undert the Java CoG Kit Public License. At this tiem the Java CoG Kit License (Section 2.3) is a simple copy of the Globus Toolkit License with the Globus Toolkit refernces being replaced by apropiate Java CoG Kit refernces and the institution, just being University of Chicago.

To collaborate with us it is best for now to just sign the Globus Toolkit contributor License and fax it to Gregor von Laszewski at 630 252 1997.

2.1 Project Registration

We wish that you to notify us about projects that you develop with the help of the Java CoG Kit. This will allow us to keep track of the use of the Java CoG Kit, as this directly affects our ability to motivate additional coding activities. Please, be so kind to send an e-mail to `gregor@mcs.anl.gov` with the subject

JAVA COG KIT USGAE

with the following additional information provided by you:

Project name:
Institution:
Main contact:
E-mail:
Web page:
Description of your project:
References:
References citing the Java CoG Kit:

In case you like to cite the Java CoG Kit in your papers, we recommend that you use the following paper:

Gregor von Laszewski, Ian Foster, Jarek Gawor, Peter Lane,
A Java Commodity Grid Kit,
Concurrency and Computation: Practice and Experience,
Pages 643-662, Volume 13, Issue 8-9, 2001.
http://www.globus.org/cog/java/

We also would like to be notified about your publications that involve the use of the Java CoG Kit, as this will help us to document its usefulness. We like to feature links to these articles, with your permission, on our Web site.

Additional references to Java CoG Kit and other Grid related activities can be found at

Some Refernces, von Laszewski : `http://www.mcs.anl.gov/˜gregor/bib`

or

Some References, Globus Project : `http://www.globus.org/research/papers.html`

.

2.2 Globus Toolkit

2.2.1 Globus Toolkit Public License (GTPL)

Globus Toolkit Public License (GTPL) Version 2

Globus Toolkit Public License Version 2, July 31, 2003

This software referred to as the Globus Toolkit software ("Software") includes voluntary contributions made to the Globus Project collaboration. Persons and entities that have made voluntary contributions are hereinafter referred to as "Contributors." This Globus Toolkit Public License is referred to herein as "the GTPL." For more information on the Globus Project, please see http://www.globus.org/.

Permission is granted for the installation, use, reproduction, modification, display, performance and redistribution of this Software, with or without modification, in source and binary forms. Permission is granted for the installation, use, reproduction, modification, display, performance and redistribution of user files, manuals, and training and demonstration slides ("Documentation") distributed with or specifically designated as distributed under the GTPL. Any exercise of rights under the GTPL is subject to the following conditions:

1. Redistributions of this Software, with or without modification, must reproduce the GTPL in: (1) the Software, or (2) the Documentation or some other similar material which is provided with the Software (if any).
2. The Documentation, alone or if included with a redistribution of the Software, must include the following notice: "This product includes material developed by the Globus Project (http://www.globus.org/)."
3. Alternatively, if that is where third-party acknowledgments normally appear, this acknowledgment must be reproduced in the Software itself.
4. Globus Toolkit and Globus Project are trademarks of the University of Chicago. Any trademarks of the University of Chicago or the University of Southern California may not be used to endorse or promote software, or products derived therefrom, and except as expressly provided herein may not be affixed to modified redistributions of this Software or Documentation except with prior written approval, obtainable at the discretion of the trademark owner from info@globus.org.
5. To the extent that patent claims licensable by the University of Southern California and/or by the University of Chicago (as Operator of Argonne National Laboratory) are necessarily infringed by the use or sale of the Software, you and your transferees are granted a non-exclusive, worldwide, royalty-free license under such patent claims, with the rights to make, use, sell, offer to sell, import and otherwise transfer the Software in source code and object code form. This patent license shall not apply to Documentation or to any other software combinations which include the Software. No hardware per se is licensed hereunder.

   If you or any subsequent transferee (a "Recipient") institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Software infringes such Recipient's patent(s), then such Recipient's rights granted under the patent license above shall terminate as of the date such litigation is filed.
6. DISCLAIMER

   SOFTWARE AND DOCUMENTATION ARE PROVIDED BY THE COPYRIGHT HOLDERS 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. THE COPYRIGHT HOLDERS AND CONTRIBUTORS MAKE NO REPRESENTATION THAT THE SOFTWARE, DOCUMENTATION, MODIFICATIONS, ENHANCEMENTS OR DERIVATIVE WORKS THEREOF, WILL NOT INFRINGE ANY PATENT, COPYRIGHT, TRADEMARK, TRADE SECRET OR OTHER PROPRIETARY RIGHT.
7. LIMITATION OF LIABILITY

   THE COPYRIGHT HOLDERS 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 OR DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

8. The Globus Project may publish revised and/or new versions of the GTPL from time to time. Each version will be given a distinguishing version number. Once Software or Documentation has been published under a particular version of the GTPL, you may always continue to use it under the terms of that version. You may also choose to use such Software or Documentation under the terms of any subsequent version of the GTPL published by the Globus Project. No one other than the Globus Project has the right to modify the terms of the GTPL.

Globus Toolkit Public License 7-31-03

2.2.2 Globus Toolkit Contributor Liceense

Grant of Licenses in Globus Toolkit Contributions, July, 2003

The undersigned licensor ("LICENSOR") has delivered or caused or permitted to be delivered to The University of Chicago, as Operator of Argonne National Laboratory, and The University of Southern California (collectively "LICENSEE") software and documentation (collectively, "CONTRIBUTIONS") created by LICENSOR or by LICENSOR's employees, associates, contractors, or collaborators. Software ("SOFTWARE") contributed by LICENSOR are source and binary software code, such as schema, bug fixes, patches, upgrades or other modifications or enhancements of the features, functionality or performance of the Globus Toolkit software ("GLOBUS TOOLKIT"). Documentation ("DOCUMENTATION") contributed by LICENSOR are print and digital media that describe and explain, such as readme files, white papers, overviews, and tutorials, and are useful with the GLOBUS TOOLKIT.

Acknowledging receipt of LICENSEE's services in distributing the GLOBUS TOOLKIT to a community of end users and developers, including LICENSOR, and LICENSEE's intention to continue such distribution, LICENSOR hereby grants to LICENSEE the following licenses (collectively, with all limitations included herein, referred to as this "Grant"):

a) a non-exclusive, worldwide, royalty-free, perpetual, transferable copyright license to install, use, reproduce, modify, display, perform, and prepare derivative works of SOFTWARE; to incorporate SOFTWARE in whole or in part and derivative works thereof into the GLOBUS TOOLKIT or into any other computer software; and to sublicense copyrights in and distribute SOFTWARE and derivative works thereof, in source and binary forms.

b) subject to termination provisions below and to the extent that patent claims licensable by LICENSOR are necessarily infringed by the use or sale of SOFTWARE alone or when combined with the GLOBUS TOOLKIT, a non-exclusive, worldwide, royalty-free license and right to sublicense under such patent claims, with the rights for LICENSEE and its sublicensees to make, use, sell, offer to sell, import and otherwise transfer the SOFTWARE in source code and object code form, alone and incorporated into the GLOBUS TOOLKIT and derivative works. This patent license shall apply to the combination of the SOFTWARE and the GLOBUS TOOLKIT if, at the time SOFTWARE is added by the LICENSEE or its transferees, such addition of the SOFTWARE causes such combination to be covered by such patent claims. The patent license shall not apply to any other combinations which include the SOFTWARE. No hardware per se is licensed hereunder.

c) a non-exclusive, worldwide, royalty-free, perpetual, transferable copyright license: to install, use, reproduce, modify, display, perform, and prepare derivative works of DOCUMENTATION; to incorporate DOCUMENTATION in whole or in part and derivative works thereof into any other documentation; and to sublicense copyrights in and distribute DOCUMENTATION and derivative works thereof.

If LICENSEE or any subsequent transferee (each referred to as a "RECIPIENT") institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the SOFTWARE or the GLOBUS TOOLKIT distributed pursuant to this Grant or the Globus ToolkitPublic License infringe such RECIPIENT's patent(s), then such RECIPIENT's rights granted under the patent license in paragraph b) above shall terminate as of the date such litigation is filed.

The foregoing licenses shall be effective so long as the terms under which LICENSEE sublicenses copyrights in and distributes the CONTRIBUTIONS effectively: (a) include compatible patent license termination language as that set forth above, and (b) disclaim liability on behalf of LICENSOR for all damages, including direct, indirect, special, incidental, exemplary and punitive damages, in substantially the same form as that included herein; and shall apply to any and all CONTRIBUTIONS specifically designated, now or in the future, as within the scope of this Grant by LICENSOR's authorized representative.

LICENSOR represents that to its knowledge it has or has obtained any and all required permissions and authority to make this Grant. Except as provided in the preceding sentence, LICENSOR PROVIDES THE CONTRIBUTIONS ON AN "AS IS" BASIS AND MAKES NO REPRESENTATIONS AND EXTENDS NO WARRANTIES OF ANY KIND, WHETHER EXPRESS OR IMPLIED, CONCERNING THE CONTRIBUTIONS. LICENSOR MAKES NO EXPRESS OR IMPLIED WARRANTIES OF MERCHANTABILITY, OF SATISFACTORY QUALITY, OR FITNESS FOR A PARTICULAR PURPOSE OR USE. LICENSOR MAKES NO REPRESENTATION THAT THE CONTRIBUTIONS WILL NOT INFRINGE ANY PATENT, COPYRIGHT, TRADE SECRET OR OTHER PROPRIETARY RIGHT. LICENSOR SHALL HAVE NO LIABILITY WITH RESPECT TO ITS OBLIGATION UNDER THIS GRANT OR OTHERWISE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, EXEMPLARY, OR PUNITIVE DAMAGES OF ANY CHARACTER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

LICENSEE shall not be required to include the name of LICENSOR or any of its employees, associates, contractors or collaborators in any copyright registration, notice or license associated with the CONTRIBUTIONS. LICENSOR hereby grants LICENSEE permission to post this Grant on a website associated with the GLOBUS TOOLKIT, provided that LICENSOR is given the opportunity to review and approve in advance any such posting; such approval not to be unreasonably withheld or delayed.

Except as expressly provided herein, nothing in this Grant shall be construed as granting any right or license under any inventions, patents, copyrights, trade secrets, or any other intellectual property rights of LICENSOR.

```
LICENSOR: _____________________________________

By:       _____________________________________

Title:    _____________________________________

Date:     _____________________________________

Address:

          _____________________________________

          _____________________________________
```

Globus Toolkit Contributor License 7-31-03

2.3 Java CoG Kit

Note that the Java CoG Kit License is just a draft and is in principle very similar to the Globus Toolkit License.

At present, we recommend to work with us under the Globus Toolkit Public License and the Globus Toolkit Contributor License.

2.3.1 Java CoG Kit Public License (JCoGPL)

Java CoG Kit Public License (JCoGPL) Draft Version

Java CoG Kit Public License Draft, February 29, 2004

Copyright 1999-2003 University of Chicago. All rights reserved.

This software referred to as the Java CoG Kit software ("Software") includes voluntary contributions made to the Globus Project collaboration. Persons and entities that have made voluntary contributions are hereinafter referred to as "Contributors." This Java CoG Kit Public License is referred to herein as "the JCoGPL." For more information on the Java CoG Kit, please see http://www.cogkits.org/.

Permission is granted for the installation, use, reproduction, modification, display, performance and redistribution of this Software, with or without modification, in source and binary forms. Permission is granted for the installation, use, reproduction, modification, display, performance and redistribution of user files, manuals, and training and demonstration slides ("Documentation") distributed with or specifically designated as distributed under the JCoGKit. Any exercise of rights under the JCoGKit is subject to the following conditions:

1. Redistributions of this Software, with or without modification, must reproduce the JCoGPL in: (1) the Software, or (2) the Documentation or some other similar material which is provided with the Software (if any).
2. The Documentation, alone or if included with a redistribution of the Software, must include the following notice: "This product includes material developed by the Globus Project (http://www.globus.org/) and the Java CoG Kit (http://www.cogkits.org)."
3. Alternatively, if that is where third-party acknowledgments normally appear, this acknowledgment must be reproduced in the Software itself.
4. Globus Toolkit Java CoG Kit and Globus Project are trademarks of the University of Chicago. Any trademarks of the University of Chicago may not be used to endorse or promote software, or products derived therefrom, and except as expressly provided herein may not be affixed to modified redistributions of this Software or Documentation except with prior written approval, obtainable at the discretion of the trademark owner from info@globus.org.
5. To the extent that patent claims licensable by the University of Chicago (as Operator of Argonne National Laboratory) are necessarily infringed by the use or sale of the Software, you and your transferees are granted a non-exclusive, worldwide, royalty-free license under such patent claims, with the rights to make, use, sell, offer to sell, import and otherwise transfer the Software in source code and object code form. This patent license shall not apply to Documentation or to any other software combinations which include the Software. No hardware per se is licensed hereunder.

   If you or any subsequent transferee (a "Recipient") institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Software infringes such Recipient's patent(s), then such Recipient's rights granted under the patent license above shall terminate as of the date such litigation is filed.
6. DISCLAIMER

   SOFTWARE AND DOCUMENTATION ARE PROVIDED BY THE COPYRIGHT HOLDERS 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. THE COPYRIGHT HOLDERS AND CONTRIBUTORS MAKE NO REPRESENTATION THAT THE SOFTWARE, DOCUMENTATION, MODIFICATIONS, ENHANCEMENTS OR DERIVATIVE WORKS THEREOF, WILL NOT INFRINGE ANY PATENT, COPYRIGHT, TRADEMARK, TRADE SECRET OR OTHER PROPRIETARY RIGHT.

7. LIMITATION OF LIABILITY

   THE COPYRIGHT HOLDERS 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 OR DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

8. The Java CoG Kit Project may publish revised and/or new versions of the JCoGPL from time to time. Each version will be given a distinguishing version number. Once Software or Documentation has been published under a particular version of the JCoGPL, you may always continue to use it under the terms of that version. You may also choose to use such Software or Documentation under the terms of any subsequent version of the JCoGPL published by the Java CoG Project. No one other than the Java CoG Project has the right to modify the terms of the JCoGPL.

Java CoG Kit Public License Draft, 29-Feb-04

2.3.2 Java CoG Kit Contributor Liceense

Grant of Licenses in Java CoG Kit Contributions, July, 2003

The undersigned licensor ("LICENSOR") has delivered or caused or permitted to be delivered to The University of Chicago, as Operator of Argonne National Laboratory (collectively "LICENSEE") software and documentation (collectively, "CONTRIBUTIONS") created by LICENSOR or by LICENSOR's employees, associates, contractors, or collaborators. Software ("SOFTWARE") contributed by LICENSOR are source and binary software code, such as schema, bug fixes, patches, upgrades or other modifications or enhancements of the features, functionality or performance of the Java CoG Kit software ("JAVA COG KIT"). Documentation ("DOCUMENTATION") contributed by LICENSOR are print and digital media that describe and explain, such as readme files, white papers, overviews, and tutorials, and are useful with the JAVA COG KIT.

Acknowledging receipt of LICENSEE's services in distributing the JAVA COG KIT a community of end users and developers, including LICENSOR, and LICENSEE's intention to continue such distribution, LICENSOR hereby grants to LICENSEE the following licenses (collectively, with all limitations included herein, referred to as this "Grant"):

a) a non-exclusive, worldwide, royalty-free, perpetual, transferable copyright license to install, use, reproduce, modify, display, perform, and prepare derivative works of SOFTWARE; to incorporate SOFTWARE in whole or in part and derivative works thereof into the JAVA COG KIT or into any other computer software; and to sublicense copyrights in and distribute SOFTWARE and derivative works thereof, in source and binary forms.

b) subject to termination provisions below and to the extent that patent claims licensable by LICENSOR are necessarily infringed by the use or sale of SOFTWARE alone or when combined with the JAVA COG KIT, a non-exclusive, worldwide, royalty-free license and right to sublicense under such patent claims, with the rights for LICENSEE and its sublicensees to make, use, sell, offer to sell, import and otherwise transfer the SOFTWARE in source code and object code form, alone and incorporated into the JAVA COG KIT and derivative works. This patent license shall apply to the combination of the SOFTWARE and the JAVA COG KIT if, at the time SOFTWARE is added by the LICENSEE or its transferees, such addition of the SOFTWARE causes such combination to be covered by such patent claims. The patent license shall not apply to any other combinations which include the SOFTWARE. No hardware per se is licensed hereunder.

c) a non-exclusive, worldwide, royalty-free, perpetual, transferable copyright license: to install, use, reproduce, modify, display, perform, and prepare derivative works of DOCUMENTATION; to incorporate DOCUMENTATION in whole or in part and derivative works thereof into any other documentation; and to sublicense copyrights in and distribute DOCUMENTATION and derivative works thereof.

If LICENSEE or any subsequent transferee (each referred to as a "RECIPIENT") institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the SOFTWARE or the JAVA COG KIT distributed pursuant to this Grant or the Java CoG KitPublic License infringe such RECIPIENT's patent(s), then such RECIPIENT's rights granted under the patent license in paragraph b) above shall terminate as of the date such litigation is filed.

The foregoing licenses shall be effective so long as the terms under which LICENSEE sublicenses copyrights in and distributes the CONTRIBUTIONS effectively: (a) include compatible patent license termination language as that set forth above, and (b) disclaim liability on behalf of LICENSOR for all damages, including direct, indirect, special, incidental, exemplary and punitive damages, in substantially the same form as that included herein; and shall apply to any and all CONTRIBUTIONS specifically designated, now or in the future, as within the scope of this Grant by LICENSOR's authorized representative.

LICENSOR represents that to its knowledge it has or has obtained any and all required permissions and authority to make this Grant. Except as provided in the preceding sentence, LICENSOR PROVIDES THE CONTRIBUTIONS ON AN "AS IS" BASIS AND MAKES NO REPRESENTATIONS AND EXTENDS NO WARRANTIES OF ANY KIND, WHETHER EXPRESS OR IMPLIED, CONCERNING THE CONTRIBUTIONS. LICENSOR MAKES NO EXPRESS OR IMPLIED WARRANTIES OF MERCHANTABILITY, OF SATISFACTORY QUALITY, OR FITNESS FOR A PARTICULAR PURPOSE OR USE. LICENSOR MAKES NO REPRESENTATION THAT THE CONTRIBUTIONS WILL NOT INFRINGE ANY PATENT, COPYRIGHT, TRADE SECRET OR OTHER PROPRIETARY RIGHT. LICENSOR SHALL HAVE NO LIABILITY WITH RESPECT TO ITS OBLIGATION UNDER THIS GRANT OR OTHERWISE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, EXEMPLARY, OR PUNITIVE DAMAGES OF ANY CHARACTER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

LICENSEE shall not be required to include the name of LICENSOR or any of its employees, associates, contractors or collaborators in any copyright registration, notice or license associated with the CONTRIBUTIONS. LICENSOR hereby grants LICENSEE permission to post this Grant on a website associated with the JAVA COG KIT, provided that LICENSOR is given the opportunity to review and approve in advance any such posting; such approval not to be unreasonably withheld or delayed.

Except as expressly provided herein, nothing in this Grant shall be construed as granting any right or license under any inventions, patents, copyrights, trade secrets, or any other intellectual property rights of LICENSOR.

```
LICENSOR: _____________________________________

By:       _____________________________________

Title:    _____________________________________

Date:     _____________________________________

Address:

          _____________________________________

          _____________________________________
```

Java CoG Kit Contributor License Draft

2.4 Other Licences

We distribute a number of other libraries with the Java CoG Kit. These libraries come with their own licences. We strongly encourage you to inspect these licenses. The can be found in the “lib” directories of the Java CoG Kit.

2.4.1 jglobus

The jglobus/lib directory contains the following licences.

jglobus : bouncycastle.LICENSE

jglobus : cryptix.LICENSE

jglobus : log4j.LICENSE

jglobus : junit.LICENSE

jglobus : puretls.LICENSE

2.4.2 ogce

The ogce/lib directory contains the following licences:

ogce : soaprmi11.LICENSE

ogce : xerces.LICENSE

ogce : xml4j.LICENSE

2.4.3 Others

[1]

2.5 GNU Public Licence

Although we prefer the development of code that is non GPL, we do have the ability to distribute components under the GNU license or are dependent on code developed with the GNU License. For thie purpose we created a seperate CVS archive in which we maintain the GPL-based code.

We will not distribute any GPL based Java CoG Kit code in binary format. It must be downloaded, compiled, and installed seperatly.

We will include more details to this issue in future. One of the Java CoG Kit Codes that will be included is the availability of a GPL based Grid shell.

At this time, we have not yet made this code available as part of the Java CoG Kit.

[1] Other licenses need to be adde here

3 Introduction

3.1 Overview

3.2 History

3.2.1 Metacomputing

3.2.2 CoG Kits

directions

developed OO

developed API based version - no success too dificult to use

Globus

API - protocol - services

Web services without modification

Inforgram first futire service

4 Installation

4.1 Download

4.2 CVS Release Tags

cvs co -r v2-0-a cog Sat Nov 15

Branch

v2-1

4.2.1 Prerequisites

In order to use the Java CoG Kit, the Java Runtime Environment version 1.4, available from the Java Web-Site is required. Additionally, if you plan to compile the Java CoG Kit from sources, you will need the full Java Development Kit, version 1.4, available from the same web-site, and a recent version of the Apache Ant build system.

At this time we recommend to use the following packages, as we have not yet tested Java CoG with any other Java version.

1. **ant-1.5.4:** The Java CoG Kit requires Apache Ant which can be downloaded form `http://ant.apache.org`.
2. JDK 1.4.2_02-b03 or above

Please note that earlier versions of the Java Development Kit contain expired root certificates, which means that you will only be able to use public key cryptography in a limited fashion.

4.3 Downloading the Java CoG Kit

Before using the Java CoG Kit, you will need to download it. At this moment, the Java CoG Kit is only available in source format and from the source repository. To download the sources from the source repository you will need to have a CVS client installed. Instructions will be provided for the command-line CVS clients (available on most UNIX and Linux machines and CYGWIN):

1. cvs -d:pserver:anonymous@cvs.globus.org:/home/dsl/cog/CVS login
2. password: Hit Enter
3. cvs -d:pserver:anonymous@cvs.globus.org:/home/dsl/cog/CVS checkout cog
4. cd cog
5. ant dist

4.4 Compiling the Java CoG Kit

In compiling the Java CoG Kit, you have two options:

1. Compiling the whole Java CoG Kit. This option may suit you if you plan to use the whole functionality of the Java CoG Kit, or if you wan to test all the features of the Java CoG Kit.
2. Compiling individual modules. This option will only compile the necessary parts needed in order to provide the particular functionality packed in a module.

4.5 Compiling the complete distribution

In the main cog directory, type: $ ant dist

A new directory named dist will be created in the cog directory. Inside the dist directory you will find a cog-<version> directory which contains libraries (lib), configuration files (etc), example files (examples) and application launchers (bin).

4.6 Compiling individual modules

The main cog directory contains a subdirectory named modules, which in turn contains all the modules that compose the Java CoG Kit. You can change directory to any of these modules and type the following in order to obtain a binary distribution directory for that module:

$ ant dist

A dist/<modulename>-<moduleversion> directory will be created containing the distribution files. Any modules that the compiled module depends on will also be compiled and included inside the same directory.

4.7 Using the Java CoG Kit

The following is the basic layout for the binary distribution directories, whether obtained by downloading the precompiled packages or by compiling the sources:

bin/ etc/ lib/

The bin directory contains launchers that can be used to start a particular application in the Java CoG Kit. The etc directory contains configuration files needed by various parts of the Java CoG Kit. The lib directory contains the jar files that belong to the Java CoG Kit, together with the libraries required to run various parts of the Java CoG Kit.

To start a particular CoG application go to the bin directory and choose its respective launcher. IMPORTANT! The Java CoG Kit now automatically detects the COG_INSTALL_PATH. If you have the variable set to a specific directory pointing to an older version of the Java CoG Kit, it may result in unexpected behavior. Please unset the COG_INSTALL_PATH variable before running any of the applications, or set it to either

cog/dist/cog-<version>

or to

cog/modules/<modulename>/dist/<modulename>-<moduleversion>

4.8 Setup

5 Contributing

5.1 Creating a module

It is easy to contribute to the Java CoG Kit through its newly designed module concept. A Sample module build file can be found in modules/template. There are a few requirements that have to be imposed in order to keep consistency:

The basic directory structure that **must** exists for each module is:

```
etc/MANIFEST.MF.head
etc/MANIFEST.MF.tail
lib/
src/
```

5.1.1 Build files

The build files for each module has 4 parts:

- build.xml : should not be modified at all unless absolutely neccessary. If there is a feature that you would like added to the build system, please tell Mike (hategan@mcs.anl.gov)
- dependencies.xml : project dependencies are stored here. Please modify it to suit your needs. An example is given in the modules/template directory
- launchers.xml : launchers that you want created in the build process. Use the example in modules/template to see how to use it
- project.properties : The module properties. The module name *must* be the same as the directory name of the module. The last line in this file contains the library dependencies for this module. If you don't add the jar files that your project requires there, it will not build. The format is a comma separated list of files. I suggest using <jar-name>.* (so that licenses and other things belonging to a jar will also be copied). Please read below about the libraries.

5.1.2 Libraries

Libraries can be found in two places:

1. `cog/lib`
2. `cog/modules/yourmodule/lib`

The build system will automatically choose the library from either of the two directories. If a library exists in both directories, priority will be given to the library in the `cog/lib` directory. This may cause your module not to build. Please talk to Gregor or Mike in this case. Also please note that the libraries in your module may at any time move to the `cog/lib` directory.

5.1.3 Source

The sources for your module. Not much to say here :)

5.1.4 Using PMD

We recommend that that developers and contributors use PMD (http://pmd.sourceforge.net) to check their code. Many of the complaints that PMD generates should be taken seriously. Still, there are instances when PMD rules do not apply for a good reason and create false positives.

To use pmd, you need to donwload it and add all its jar files to the pmd directory. Afterwards, just run 'ant pmd' in the module you want to check. It will generate both an on-screen report and an html report (pmd-report.html)

5.1.5 Documenting the modules

README TODO CHANGES PMD

5.1.6 Maintaining a module

5.1.7 Launchers

5.1.8 Webstart

5.2 Coding Guidelines for the Java CoG Kit

The Java CoG Kit follows in general the basic coding conventions given in the "Sun Coding Conventions for the Java Programming Language" (http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html). Additionally we have the following rules.

5.2.1 Imports

All imports must be single class and explicit. I.e. import <package>.* is not allowed.

5.2.2 Indentation

All indentation levels should be 4 spaces. No editor tabs are allowed unless they are converted to 4 spaces before saving the file.

5.2.3 Brackets

In contrast to the OGSA coding guides, we only allow the use of brackets as defined in the Java Coding guidelines. E.g.

```
for (index = 0; index < length; index++) {
    <code>
}
```

5.2.4 Variables

No acronyms or abbreviations should be used. E.g. a = b + mVarLen should be avoided and instead use: totalLength = partLength + newLength

5.2.5 Instance Variables

Use “this.” prefix when referencing instance variables, e.g.:

```
public MyClass (ServicePropertiesInterface properties) {
    this.properties = properties;
}

public int foo () {
    int localInt = 3;
    return this.instanceInt + localInt;
}
```

5.3 One-Liners

Even single line statements should be inside brackets. E.g.

```
if (isEmpty) {
    return;
}
```

5.3.1 Logging

Log4J should be used exclusively. System.out/err.println is not allowed. Further, exceptions should be logged.

5.3.2 Testing

Each component/class should have a JUnit test The tests should be put in test/ directory under each package directory.

5.3.3 Internationalization

The core framework should be fully internatioanlized. The samples may be internationalized. The Java I18n/L10n Toolkit may be used to verify whether code is international.

5.3.4 Library Reuse

Treat all code as a library, and as a reusable component. Calls to System.exit() are disallowed (except the main method)

5.3.5 Exceptions

Use chained exceptions. Java CoG Kit provides two simple generic exception classes for chaining multiple exceptions together. Look at ChainedException and ChainedIOException.

6 Modules

6.1 util

6.2 certrequest

7 Core

7.1 Introduction

The Java Cog Kit 2.0 core[1] (cog-core) is an add on to the cog-jglobus library. It includes many advanced features to make Grid programming easier. The core module provides an abstraction layer for various low level Grid implementations such as Globus Toolkit v2 and v3. A Grid application developer can port Grid applications from one implementation (GT2) to another (GT3) by simply changing the underlying implementation provider. Thus, all applications developed using the cog-core APIs are compatible with all the underlying Grid implementations supported by cog-core. The current version of the core module provides support for GT2, GT3, and SSH implementations. Other platforms will be supported based on availability of resources.

Further cog-core also provides several constructs whereby simple execution dependencies (workflows) can be expressed as a directed acyclic graph (DAG) or hierarchical DAG where each Grid task can interface with a different Grid implementation. For more sophisticated workflow functionality the reader is directed to the Java CoG Kit Karajan module in Chapter 8

Hence, cog-core offers the following benefits:

- develop client applications that will be inter-operable across multiple Grid backend implementations;
- provide re-usable code to support rapid prototyping of basic Grid access patterns;
- provide an open-source and extensible architecture that can be built collectively and incrementally based on community feedback; and
- access the same set of interfaces implemented in disparate technologies.

7.2 Installation

7.2.1 Download

The Java CoG Kit 2.0 core module can be downloaded from the Java CoG Kit CVS archive. Instructions regarding the Java CoG Kit requirements and details on obtaining the Java CoG Kit sources, are available in Section 4.

It must be noted that cog-core is explicitly a client-side library. Current version of cog-core provides support for GT2, GT3, and SSH. Hence, in order to execute tasks against these implementations, the reader is directed to install GT2.4, GT3.0.3, and SSH server. For further details on installing the Globus Toolkit please visit the Globus Alliance webpage `http://www.globus.org`

[1] Formerly known as the GridSDK module

7.2.2 Compile

To compile the core module, change directory to *cog*/modules/core and type 'ant dist'. This will compile cog-core and all its dependencies. It will also create a *dist* directory containing the distribution of the core module. Inside the *dist* directory, the *bin* directory will contain the necessary scripts that can be used to launch several commandline clients and example applications.

7.2.3 Configuration

Cog-core can be configured via the cog-core.properties file in the /.globus directory. The user can set the following properties in this configuration file:

```
#provider    class
GT2 = org.globus.cog.core.impl.gt2
GT3 = org.globus.cog.core.impl.gt3
SSH = org.globus.cog.core.impl.ssh
SCHEMA_LOCATION = <COG_HOME>/modules/core/schema
```

GT2, GT3, and SSH are default providers. SCHEMA_LOCATION indicates the directory location of the schemas required by GT3. and `<COG_HOME>` points to the location where the CoG is installed.

Hence, an example cog-core.properties file would be similar to:

```
#Java CoG Kit Core module
GT2 = org.globus.cog.core.impl.gt2
GT3 = org.globus.cog.core.impl.gt3
SSH = org.globus.cog.core.impl.ssh
SCHEMA_LOCATION = /home/user-name/cog/core/modules/core/schema
```

7.2.4 Examples

Several examples that demonstrate the ease of use and functionality of the Java CoG Kit Core are provided. These examples are available in the `../modules/core/src/org/globus/cog/core/examples` directory.

The examples are further divided into the following packages:

gt2 : showcasing the gt2 functionality.

1. The GT2 JobSubmission example demonstrates the ability to submit to a PBS batch queue.
2. The GT2 FileTransfer example demonstrates the ability to perform a third party file transfer using Grid FTP.

gt3 : showcasing the gt3 functionality.

1. The GT3 JobSubmission example demonstrates the ability to submit to a MasterForkManagedJobFactory service. It can also be used for PBS managed factory services.
2. The GT3 FileTransfer includes examples for third party transfers for single as well as multiple files. It uses the MultiRFT Grid service.

ssh : showcasing the ssh functionality.

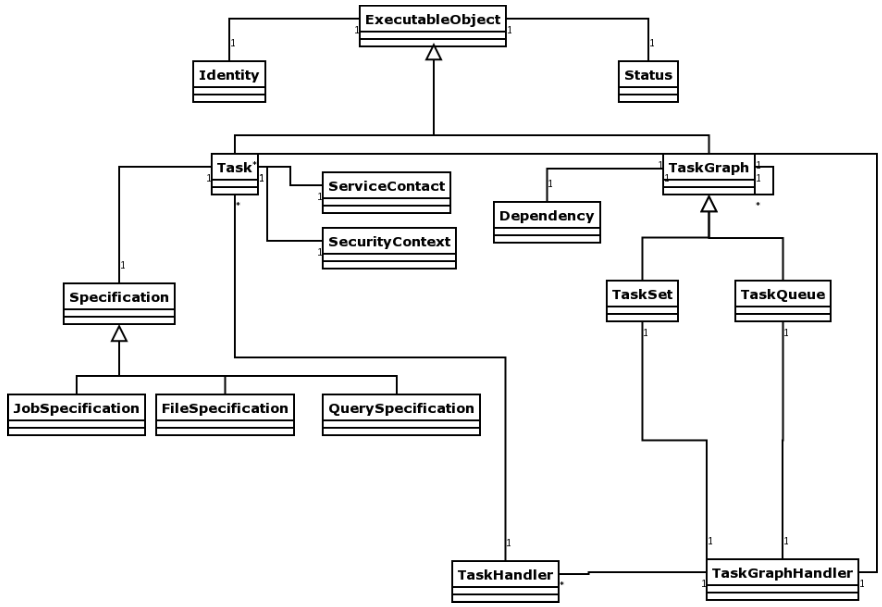


Figure 7.1: Core UML Class Diagram

1. The SSH JobSubmission and FileTransfer examples demonstrate the ability to use SSH and SSH credentials to perform the corresponding tasks.

misc : demonstrating the combination of the gt2, gt3, and ssh platforms with execution dependencies. The examples in misc package show how to create a directed acyclic graph (DAG) and visualize it using the grapheditor module. It also shows how to create a hierarchical DAG and visualize it.

After successfully compiling the core module, these examples can be executed from the launcher scripts available in `../modules/core/dist/bin` directory.

7.3 Design

One of the most important usage patterns in Grid computing is the execution of a Grid task. An extension to this basic Grid execution pattern is a Grid workflow pattern that enable the user to submit a set of Grid tasks along with an execution dependency. Therefore, the initial design of cog-core concentrates on providing the artifacts required to support these important usage patterns. Other Grid patterns can be supported by extending the flexible cog-core design based on community feedback.

Figure 7.1 shows the class diagram of cog-core. A detailed listing of the attributes and functions for each class has been omitted for simplicity. In the rest of this section we describe the important entities designed and their semantics as a part of the offered functionality.

7.3.1 ExecutableObject

An *ExecutableObject* provides a high level abstraction for artifacts that can be executed on the Grid. It can be specialized as a Grid Task or a TaskGraph. An ExecutableObject in cog-core has a unique identity and an execution status.

Listing 7.1: Interface definition for ExecutableObject

```
public interface ExecutableObject
{
    public static final int TASK = 1;
    public static final int TASKGRAPH = 2;

    public void setName(String name);
    public String getName();

    public void setIdentity(Identity id);
    public Identity getIdentity();

    public int getObjectType();

    public void setStatus(Status status);
    public void setStatus(int status);
    public Status getStatus();
}
```

7.3.2 Task

A *Task* is the atomic unit of execution in cog-core. It represents a generic Grid functionality including remote job execution, file transfer request, or information query. It extends the ExecutableObject, hence it has a unique identity and execution status. It also has a security context, a specification, and a service contact.

The task identity helps in uniquely representing the task across the Grid. The security context represents the abstract security credentials of the task. It is apparent that every underlying Grid implementation enforces its own security requirements therefore making it necessary to abstract a generalized security context. Hence, the security context in cog-core offers a common construct that can be extended by the different implementations of Grid to satisfy the corresponding backend requirements. The task specification represents the actual attributes or parameters required for the execution of the Grid-centric task. The generalized specification can be extended for common Grid tasks such as remote job execution, file transfer, and information query. The service contact associated with a task symbolizes the Grid resource required to execute it.

Listing 7.2: Interface definition for Task

```
public interface Task extends ExecutableObject
{
    public static final int JOB_SUBMISSION = 1;
    public static final int FILE_TRANSFER = 2;
    public static final int INFORMATION_QUERY = 3;

    public void setType(int type);
    public int getType();

    public void setProvider(String provider);
```

```
    public String getProvider();

    public void setSpecification(Specification specification);
    public Specification getSpecification();

    public void setSecurityContext(SecurityContext security);
    public SecurityContext getSecurityContext();

    public void setServiceContact(ServiceContact servicecontact);
    public ServiceContact getServiceContact();

    public void setStdOutput(String output);
    public String getStdOutput();

    public void setStdError(String error);
    public String getStdError();

    public void setAttribute(String name, Object value);
    public Object getAttribute(String name);

    public void addStatusListener(StatusListener listener);
    public void removeStatusListener(StatusListener listener);

    public void addOutputListener(OutputListener listener);
    public void removeOutputListener(OutputListener listener);

    public void fromXML(String task);
    public String toXML();
    public void fromString(String task);
    public String toString();

    public boolean isUnsubmitted();
    public boolean isActive();
    public boolean isCompleted();
    public boolean isSuspended();
    public boolean isFailed();
    public boolean isCanceled();

    public Calendar getSubmittedTime();
    public Calendar getCompletedTime();
}
```

7.3.3 Specification

Every Grid Task has an associated *Specification* that dictates the objective of the task and the environment required to achieve the objective. The TaskHandler manage the tasks based on the parameters specified in the task specification.

Listing 7.3: Interface definition for Specification

```
public interface Specification
{
    public static final int JOB_SUBMISSION = 1;
    public static final int FILE_TRANSFER = 2;
    public static final int INFORMATION_QUERY = 3;
```

```
    public void setType(int type);
    public int getType();

    public void setSpecification(String specification);
    public String getSpecification();
}
```

A task specification is a generalized concept and can be further categorized into JobSpecification, FileSpecification, and QuerySpecification (not implemented at this time). It must be noted that the specific parameters required in a task specification depend on the underlying Grid implementation used for the execution of the Task. For example, GT3 has several required parameters that are not supported by GT2 (and vice versa). However, the specification classes in cog-core offer some commonly used attributes which can be extended or omitted based on the requirements of the task and specific Grid implementation.

The JobSpecification mentions all the important attributes needed for the remote job execution. Most of the attributes provided by the JobSpecification class are similar to the ones available in the Resource Specification Language (RSL) supported by the Globus Toolkit. Nevertheless, additional attributes can be added based on specific requirements.

Listing 7.4: Interface definition for JobSpecification

```
public interface JobSpecification extends Specification
{
    public void setExecutable(String executable);
    public String getExecutable();

    public void setDirectory(String directory);
    public String getDirectory();

    public void setArguments(String arguments);
    public String getArguments();

    public void setStdOutput(String output);
    public String getStdOutput();

    public void setStdInput(String input);
    public String getStdInput();

    public void setStdError(String error);
    public String getStdError();

    public void setCount(int count);
    public Integer getCount();

    public void setBatchJob(boolean bool);
    public boolean isBatchJob();

    public void setRedirected(boolean bool);
    public boolean isRedirected();

    public void setLocalExecutable(boolean bool);
    public boolean isLocalExecutable();

    public void setAttribute(String name, String value);
```

```
    public String getAttribute(String name);
    public Enumeration getAllAttributes();
}
```

The *FileSpecification* provides the commonly used attributes for file transfers between Grid resources. It must be noted once again that not all attributes are supported by every Grid implementation.

Listing 7.5: Interface definition for FileSpecification

```
public interface FileSpecification extends Specification
{
    public void setSourceServer(String server);
    public String getSourceServer();

    public void setDestinationServer(String server);
    public String getDestinationServer();

    public void setSourceDirectory(String directory);
    public String getSourceDirectory();

    public void setDestinationDirectory(String directory);
    public String getDestinationDirectory();

    public void setSourceFile(String file);
    public String getSourceFile();

    public void setDestinationFile(String file);
    public String getDestinationFile();

    public void setSource(String source);
    public String getSource();

    public void setDestination(String destination);
    public String getDestination();

    public void setBinary(boolean bool);
    public boolean isBinary();

    public void setNotpt(boolean bool);
    public boolean isNotpt();

    public void setDcau(boolean bool);
    public boolean isDcau();

    public void setBlockSize(int size);
    public int getBlockSize();

    public void setTcpBufferSize(int size);
    public int getTcpBufferSize();
    public void setParallelStreams(int value);
    public int getParallelStreams();

    public void setThirdParty(boolean bool);
    public boolean isThirdParty();

    public void setAttribute(String name, Object value);
```

```
    public Object getAttribute(String name);
}
```

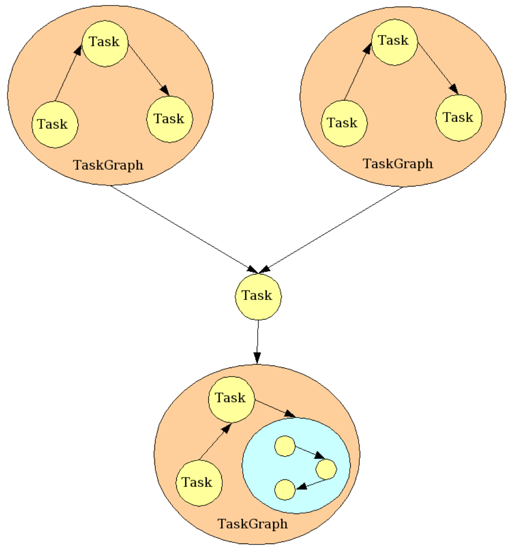


Figure 7.2: A TaskGraph can represent multiple levels of hierarchical DAG

7.3.4 TaskGraph

A *TaskGraph* provides a building block for expressing complex dependencies between tasks. All significantly advanced applications require mechanisms to execute client-side workflows that process the tasks based on user-defined dependencies. Hence, the data structure representing the TaskGraph aggregates a set of ExecutableObjects (Tasks and TaskGraphs) and allows the user to define dependencies between these tasks. In graph theoretical terms, a TaskGraph provides the artifacts to express workflows as a hierarchical directed acyclic graph (see Figure 7.2). A TaskGraph can theoretically contain infinite levels of hierarchy. However, practically it is constrained with the availability of resources (memory) on a particular system.

Listing 7.6: Interface definition for TaskGraph

```
public interface TaskGraph extends ExecutableObject
{
    public void add(ExecutableObject graphNode);
    public ExecutableObject remove(Identity id);
    public ExecutableObject get(Identity id);

    public ExecutableObject[] toArray();
    public Enumeration elements();

    public void setDependency(Dependency dependency);
    public Dependency getDependency();
```

```
    public void addDependency(Identity from, Identity to);
    public boolean removeDependency(Identity from, Identity to);

    public void setAttribute(String name, Object value);
    public Object getAttribute(String name);

    public void addStatusListener(StatusListener listener);
    public void removeStatusListener(StatusListener listener);

    public int getSize();
    public boolean isEmpty();
    public boolean contains(Identity id);

    public int getUnsubmittedCount();
    public int getSubmittedCount();
    public int getActiveCount();
    public int getCompletedCount();
    public int getSuspendedCount();
    public int getResumedCount();
    public int getFailedCount();
    public int getCanceledCount();
}
```

Cog-core provides two additional utility classes that specializes the functionality of the TaskGraph. The *TaskSet* is a special type of TaskGraph with no dependencies. Intuitively, it represents a bag of tasks that can be executed in parallel. The *TaskQueue* is another specialized TaskGraph that represents a first-in-first-out (FIFO) queue. The dependencies in a TaskQueue are not set explicitly, but are maintained implicitly based on the addition of a Task to the TaskQueue.

7.3.5 Status

Every ExecutableObject (Task or TaskGraph) has an associated execution status. An ExecutableObject can be in one of the following status: unsubmitted, submitted, active, suspended, resumed, failed, canceled, and completed. It must be noted that not all status' are supported by every Grid implementation. In other words, for some Grid implementations it may not be possible to suspend and resume remote execution.

It is easy to associate a simple Task with one of the above mentioned status. For example, initially the task is unsubmitted; its status changes to submitted when it is handled by a handler; its status changes to active when it is being executed remotely, and so on. However, it is not apparent as to how a TaskGraph is mapped to one the supported status. Cog-core uses the following logic to map a TaskGraph to its appropriate status.

Listing 7.7: Pseudocode to determine the status of a TaskGraph

```
if (any Task in the TaskGraph has failed)
{
  status = failed
}
else if (all tasks are unsubmitted)
{
  status = unsubmitted
}
```

```
else if (every task is either completed or canceled)
{
  status = completed
}
else if (any task is suspended)
{
  status = suspended
}
else if (any task is either active or resumed)
{
  status = active
}
else if (any task is submitted)
{
  status = submitted
}
else
{
  impossible to get here, the above cases take care
  of all conditions.
}
```

Listing 7.8: Interface definition for Status

```
public interface Status
{
    public static final int UNSUBMITTED = 0;
    public static final int SUBMITTED = 1;
    public static final int ACTIVE = 2;
    public static final int SUSPENDED = 3;
    public static final int RESUMED = 4;
    public static final int FAILED = 5;
    public static final int CANCELED = 6;
    public static final int COMPLETED = 7;

    public abstract void setStatusCode(int status);
    public abstract int getStatusCode();
    public abstract void setPrevStatusCode(int status);
    public abstract int getPrevStatusCode();
    public abstract void setException(Exception exception);
    public abstract Exception getException();
    public abstract void setMessage(String message);
    public abstract String getMessage();
    public void setTime(Calendar time);
    public Calendar getTime();
}
```

7.3.6 Handlers

Cog-core contains the *TaskHandler* and the *TaskGraphHandler*, to process a Task and a TaskGraph respectively. Once a Task or a TaskGraph is submitted to the appropriate handler, the handler interacts with the desired Grid implementation and accomplishes the necessary tasks. The handlers in cog-core can be viewed as adaptors that translate the abstract definitions of a Task and TaskGraph into implementation specific constructs that is understood by the backend Grid services. For example, a GT3 TaskHandler will extract the appropriate attributes from the cog-core Task and make the necessary calls to the remote Grid service factory, retrieve the Grid service handle, and interact with the newly created service instance. Symmetric translations would be done for other Grid implementations. Intuitively, a Handler is specific to the backed implementation and is the only part of cog-core that needs to be extended for supporting additional Grid implementations. Since cog-core supports GT2, GT3, and SSH the appropriate handlers for these are available. For cog-core to support Unicore, all one needs to do is to add a Unicore handler.

The TaskHandler provides a simple interface to handle a generic Grid task submitted to it. It is capable of categorizing the tasks and providing the appropriate functionality for it. For example, the task handler will handle a remote job execution differently than a file transfer request. Cog-core does not impose any restrictions on the implementation of the task handler as long as its working is transparent to the end user.

Listing 7.9: Interface definition for TaskHandler

```
public interface TaskHandler
{
    public static final int GENERIC = 1;
    public static final int GT2 = 2;
    public static final int GT3 = 3;


    public void setType(int type);
    public int getType();

    public void submit(Task task)
        throws
            IllegalSpecException,
            InvalidSecurityContextException,
            InvalidServiceContactException,
            TaskSubmissionException;

    public void suspend(Task task)
        throws InvalidSecurityContextException,
        TaskSubmissionException;

    public void resume(Task task)
        throws InvalidSecurityContextException,
        TaskSubmissionException;

    public void cancel(Task task)
        throws InvalidSecurityContextException,
        TaskSubmissionException;

    public void remove(Task task)
        throws ActiveTaskException;
    public Task[] getAllTasks();
    public Enumeration getActiveTasks();
    public Enumeration getFailedTasks();
    public Enumeration getCompletedTasks();
    public Enumeration getSuspendedTasks();
    public Enumeration getResumedTasks();
    public Enumeration getCanceledTasks();
}
```

The TaskGraphHandler provides a similar functionality as the task handler interface. However, it has an additional responsibility of enforcing the dependency on the graph-like task sets submitted to it. It can be implemented as an advanced workflow engine coordinating the execution of tasks on corresponding Grid resources honoring the user-defined dependencies.

Listing 7.10: Interface definition for TaskGraphHandler

```
public interface TaskGraphHandler
{
    public void submit(TaskGraph taskgraph)
       throws
       IllegalSpecException,
       InvalidSecurityContextException,
       InvalidServiceContactException,
       TaskSubmissionException;

    public void suspend()
       throws InvalidSecurityContextException,
       TaskSubmissionException;

    public void resume()
       throws InvalidSecurityContextException,
       TaskSubmissionException;

    public void cancel()
       throws InvalidSecurityContextException,
       TaskSubmissionException;

    public Task[] getAllTask();
    public Enumeration getActiveTasks();
    public Enumeration getFailedTasks();
    public Enumeration getCompletedTasks();
    public Enumeration getSuspendedTasks();
    public Enumeration getResumedTasks();
    public Enumeration getCanceledTasks();
}
```

7.4 Programmer's Guide

1. Executing a remote job execution task (7.4.1)
2. Executing a third party file transfer task (7.4.2)
3. Executing a simple TaskGraph (DAG) (7.4.3)
4. Executing a hierarchical DAG (7.4.4)
5. Writing a custom TaskHandler (7.4.5)

7.4.1 Executing a remote job execution task

Executing a remote job becomes extremely simple with cog-core. To begin with, create a Task with the appropriate attributes.

Listing 7.11: Create a Task object

```
/* Create a new job submission task named ``myTestTask'' */
```

```
Task task = new TaskImpl(``myTestTask'', Task.JOB_SUBMISSION);

/* Set the desired provider. Default options are
   GT2, GT3, or SSH
*/
task.setProvider(``GT3'');
```

Then, create a JobSpecification for the task and set the appropriate attributes as per the task requirements.

Listing 7.12: Create a task specification

```
/* Create a new JobSpecification */
JobSpecification spec = new JobSpecificationImpl();


/* Set the location and name of the executable.
   If the executable is a local executable, then
   spec.setLocalExecutable(true)
*/
spec.setExecutable(``/bin/ls'');


/* Set the arguments (if any)
   for the executable
*/
spec.setArguments(``-la'');


/* Set the name of the file which serves
   as the input to the executable

   If the input file needs to be redirected
   from the local machine, then
   spec.setLocalExecutable(true)
*/
spec.setStdInput(``core-testInput'');


/* Set the name of the file to which the remote
   output must be stored in.

   If the remote output needs to be redirected
   to the local machine, then
   spec.setRedirected(true)

   If the remote output needs to be manipulated at
   the local machine rather than storing it in a
   file, then
   spec.setRedirected(true);
   spec.setStdOutput(null);
   The output is now available from
   task.getOutput(); and can be used
   or displayed as desired.
*/
spec.setStdOutput(``core-testOutput'');
```

```
/* Set the execution mode of the job */
spec.setBatchJob(true);


/* Add additional attributes that are not
   provided by default. These add on
   attributes will be considered by the
   handler only if it supports it.
*/
spec.setAttribute(``runCount'',``546'');


/* Assign this specification to the task */
task.setSpecification(spec);
```

Next, assign the desired security credentials to the task. This step assumes you have a valid uses certificate successfully obtained from appropriate certificate authority.

Listing 7.13: Create security credentials

```
/* Since the provider is GT3
   create a GlobusSecurityContext.

   If a non-globus security context is
   required, then use the
   SecurityContextImpl class and set the
   credentials as required by the handler
*/
GlobusSecurityContextImpl securityContext =
    new GlobusSecurityContextImpl();


/* Assign the default credentials
   available as a valid proxy certificate
   whose location is specified in the
   cog.properties file present in the
   \$HOME/.globus directory

   To assign non-default credentials
   create a GSSCredential and pass
   this GSSCredential as the argument
   instead of null
*/
securityContext.setCredentials(null);


/* Assign this security credential to the task */
task.setSecurityContext(securityContext);
```

Next, assign a ServiceContact to the task. This attribute defines the location of the remote Grid resource where the task is to be executed.

Listing 7.14: Create a service contact

```
ServiceContact service =
    new ServiceContactImpl(
```

```
    ``http://127.0.0.1:8080/
    ogsa/services/base/gram/
    MasterForkManagedJobFactoryService'');

task.setServiceContact(service);
```

Next, create a TaskHandler and submit the task for execution.

Listing 7.15: Create a task handler

```
TaskHandler handler = new TaskHandlerImpl();
    try
    {
      handler.submit(task);
    } catch (InvalidSecurityContextException ise)
    {
      logger.error(``Security Exception'');
      ise.printStackTrace();
      System.exit(1);
    } catch (TaskSubmissionException tse)
    {
      logger.error(``TaskSubmission Exception'');
      tse.printStackTrace();
      System.exit(1);
    } catch (IllegalSpecException ispe)
    {
      logger.error(``Specification Exception'');
      ispe.printStackTrace();
      System.exit(1);
    } catch (InvalidServiceContactException isce)
    {
      logger.error(``Service Contact Exception'');
      isce.printStackTrace();
      System.exit(1);
    }
```

If it is required to monitor the status of the task (desired in most interactive tasks), then before submitting the task to a handler subscribe to the task for its status changes.

```
task.addStatusListener(this);
```

If registered to listen to the status notification of the task, implement the statusChanged() function.

```
public void statusChanged(StatusEvent event)
{
    Status status = event.getStatus();

    logger.debug(``Status changed to ''
        + status.getStatusCode());

    if (status.getStatusCode() == Status.COMPLETED)
    {
      /* Makes sense if
         spec.setRedirected(true);
     spec.setStdOutput(null);
```

```
        */
        logger.debug(``Output = ''
            + task.getStdOutput());
        System.exit(1);
    }
}
```

7.4.2 Executing a third party file transfer task

Executing a file transfer is extremely simple with cog-core. To begin with, create a Task with the appropriate attributes.

Listing 7.16: Create a Task object

```
/* Create a new file transfer task named ``myTestTask'' */
Task task = new TaskImpl(``myTestTask'', Task.FILE_TRANSFER);

/* Set the desired provider. Default options are
   GT2, GT3, or SSH
*/
task.setProvider(``GT2'');
```

Then, create a FileSpecification for the task and set the appropriate attributes as per the task requirements.

Listing 7.17: Create a task specification

```
/* Create a new FileSpecification */
FileSpecification spec = new FileSpecificationImpl();

/* Set the source and destination files */
spec.setSource(``gsiftp://domain:2811//home/filename'');
spec.setDestination(``gsiftp://domain:2811//home/filename'');

/* If it is a third party file transfer */
spec.setThirdParyt(true);

/* Assign this specification to the task */
task.setSpecification(spec);
```

Next, assign the desired security credentials to the task. This step assumes you have a valid uses certificate successfully obtained from appropriate certificate authority.

Listing 7.18: Create security credentials

```
/* Since the provider is GT2
   create a GlobusSecurityContext.

   If a non-globus security context is
   required, then use the
   SecurityContextImpl class and set the
   credentials as required by the handler
*/
GlobusSecurityContextImpl securityContext =
    new GlobusSecurityContextImpl();
```

```
/* Assign the default credentials
   available as a valid proxy certificate
   whose location is specified in the
   cog.properties file present in the
   \$HOME/.globus directory

   To assign non-default credentials
   create a GSSCredential and pass
   this GSSCredential as the argument
   instead of null
*/
securityContext.setCredentials(null);


/* Assign this security credential to the task */
task.setSecurityContext(securityContext);
```

Note that for GT2 file transfers, there is no need to assign a ServiceContact since the source and destination file names implicitly contain the remote machine names. However, for other providers it may be required to specify the ServiceContact.

Listing 7.19: Create a service contact

```
ServiceContact service =
    new ServiceContactImpl(``127.0.0.1'');

task.setServiceContact(service);
```

Next, create a TaskHandler and submit the task for execution.

Listing 7.20: Create a task handler

```
TaskHandler handler = new TaskHandlerImpl();
    try
    {
      handler.submit(task);
    } catch (InvalidSecurityContextException ise)
    {
      logger.error(``Security Exception'');
      ise.printStackTrace();
      System.exit(1);
    } catch (TaskSubmissionException tse)
    {
      logger.error(``TaskSubmission Exception'');
      tse.printStackTrace();
      System.exit(1);
    } catch (IllegalSpecException ispe)
    {
      logger.error(``Specification Exception'');
      ispe.printStackTrace();
      System.exit(1);
    } catch (InvalidServiceContactException isce)
    {
      logger.error(``Service Contact Exception'');
      isce.printStackTrace();
      System.exit(1);
    }
```

If it is required to monitor the status of the task (desired in most interactive tasks), then before submitting the task to a handler subscribe to the task for its status changes.

```
task.addStatusListener(this);
```

If registered to listen to the status notification of the task, implement the statusChanged() function.

```
public void statusChanged(StatusEvent event)
{
    Status status = event.getStatus();

    logger.debug(``Status changed to ''
       + status.getStatusCode());

    if (status.getStatusCode() == Status.COMPLETED ||
        status.getStatusCode() == Status.FAILED)
    {
      logger.info(``Task Done'');
      System.exit(1);
    }
}
```

7.4.3 Executing a simple TaskGraph (DAG)

In order to create a TaskGraph, we assume that we have created 3 tasks: task1, task2, and task3. Instructions for creating job submission and file transfer tasks are available in the previous sections (7.4.1 and 7.4.2). We then create a TaskGraph and add a dependency between these tasks.

Listing 7.21: Create a TaskGraph with a dependency

```
TaskGraph tg = new TaskGraphImpl();


/* Give a convenient name to the TaskGraph */
tg.setName(``testGraph'');


/* Add the tasks to the TaskGraph */
tg.add(task1);
tg.add(task2);
tg.add(task3);


/* Add dependencies between these tasks.

   Dependency is added as
   task1 --> task2 --> task3.

   This implies task1 is executed before task2
   and task2 is executed before task3.
*/
tg.addDependency(task1.getIdentity(),
                 task2.getIdentity());
```

```
tg.addDependency(task2.getIdentity(),
                 task3.getIdentity());
```

Next, create a TaskGraphHandler and submit the task for execution.

Listing 7.22: Create a task graph handler

```
TaskGraphHandler handler = new TaskGraphHandlerImpl();
    try
    {
      handler.submit(tg);
    } catch (InvalidSecurityContextException ise)
    {
      logger.error(``Security Exception'');
      ise.printStackTrace();
      System.exit(1);
    } catch (TaskSubmissionException tse)
    {
      logger.error(``TaskSubmission Exception'');
      tse.printStackTrace();
      System.exit(1);
    } catch (IllegalSpecException ispe)
    {
      logger.error(``Specification Exception'');
      ispe.printStackTrace();
      System.exit(1);
    } catch (InvalidServiceContactException isce)
    {
      logger.error(``Service Contact Exception'');
      isce.printStackTrace();
      System.exit(1);
    }
```

If it is required to monitor the status of the task graph (desired in most interactive task graphs), then before submitting the task graph to a handler subscribe to the task graph for its status changes.

```
tg.addStatusListener(this);
```

If registered to listen to the status notification of the task graph, implement the statusChanged() function.

```
public void statusChanged(StatusEvent event)
{
    Status status = event.getStatus();

    logger.debug(``Status changed to ''
       + status.getStatusCode());

    if (status.getStatusCode() == Status.COMPLETED ||
        status.getStatusCode() == Status.FAILED)
    {
      logger.info(``Task Graph Done'');
      System.exit(1);
    }
}
```

7.4.4 Executing a hierarchical TaskGraph

In order to create a hierarchical TaskGraph, we assume that we have created 3 tasks and 1 TaskGraph: task1, task2, task3, and tg. Instructions for creating job submission and file transfer tasks and simple TaskGraphs are available in the previous sections (7.4.1, 7.4.2, and 7.4.3). We then create a TaskGraph and add a dependency between these ExecutableObjects.

Listing 7.23: Create a TaskGraph with a dependency

```
TaskGraph htg = new TaskGraphImpl();


/* Give a convenient name to the TaskGraph */
htg.setName(``testGraph'');


/* Add the ExecutableObjects to the TaskGraph */
htg.add(task1);
htg.add(task2);
htg.add(task3);
htg.add(tg);


/* Add dependencies between these ExecutableObjects.

   Dependency is added as
   task1 --> task2 --> task3 --> tg.

   This implies task1 is executed before task2
   and task2 is executed before task3, and
   task3 is executed before TaskGraph tg.
*/
htg.addDependency(task1.getIdentity(),
                  task2.getIdentity());

htg.addDependency(task2.getIdentity(),
                  task3.getIdentity());

htg.addDependency(task3.getIdentity(),
                  tg.getIdentity());
```

Next, create a TaskGraphHandler and submit the task for execution.

Listing 7.24: Create a task graph handler

```
TaskGraphHandler handler = new TaskGraphHandlerImpl();
    try
    {
      handler.submit(htg);
    } catch (InvalidSecurityContextException ise)
    {
      logger.error(``Security Exception'');
      ise.printStackTrace();
      System.exit(1);
    } catch (TaskSubmissionException tse)
    {
      logger.error(``TaskSubmission Exception'');
```

```
      tse.printStackTrace();
      System.exit(1);
    } catch (IllegalSpecException ispe)
    {
      logger.error(``Specification Exception'');
      ispe.printStackTrace();
      System.exit(1);
    } catch (InvalidServiceContactException isce)
    {
      logger.error(``Service Contact Exception'');
      isce.printStackTrace();
      System.exit(1);
    }
```

If it is required to monitor the status of the task graph (desired in most interactive task graphs), then before submitting the task graph to a handler subscribe to the task graph for its status changes.

```
htg.addStatusListener(this);
```

If registered to listen to the status notification of the task graph, implement the statusChanged() function.

```
public void statusChanged(StatusEvent event)
{
    Status status = event.getStatus();

    logger.debug(``Status changed to ''
       + status.getStatusCode());

    if (status.getStatusCode() == Status.COMPLETED ||
        status.getStatusCode() == Status.FAILED)
    {
      logger.info(``Task Graph Done'');
      System.exit(1);
    }
}
```

7.4.5 Writing a custom TaskHandler

To write a custom TaskHandler, create a class, say foo.bar.MyHandler that implements the org.globus.cog.core.interfaces.TaskHandler interface.

In order to successfully execute an ExecutableObject with this custom handler you need to associate this handler with a provider name, say "MyProvider". And provide the mapping between the provider name and the class name using the cog.properties file. Instructions for adding an entry in the cog.properties file is available in Section 7.2.3. Hence, we add the following entry in the cog.properties file:

```
MyProvider = foo.bar.MyHandler
```

Now, in order to use this handler with any ExecutableObject simple associate that ExecutableObject with the provider "MyProvider".

```
task.setProvider(``MyProvider'');
```

7.5 jglobus

8 Karajan

Karajan is a workflow language and workflow engine. It aims to provide the scientific community with an easy to use tool to define complex jobs on computational grids, while keeping scalability and offering some advanced features, like failure handling, checkpointing, dynamic workflows, and distributed workflows.

Workflows in Karajan are defined using a structured language based on XML, and extensible through Java. The building block of the language is the element, which loosely translates into an XML element/container. Various elements are included, such as elements for parallel processing, parallel iterators, grid elements (ie. job submission and file transfer), etc. Common tasks can be grouped using templates, and reused from multiple locations.

The execution engine in Karajan is based on an event model, which allows effective separation between the workflow specification and the runtime state. Elements react to events received from other elements, and generate their own events. These events provide notification of status changes within the execution, or can be used to control the execution of elements. The complete runtime state is contained within the events, which allows the elements themselves to exist on different resources. This mechanism also allows an external controller, which has access to these events, to completely control the execution of the worklfow. It also allows a certain level of modification to the elements to be performed, at runtime, without affecting the execution of other elements.

As an example, suppose a large job requires a transfer of the resulting data, after the completion of all calculations. Also suppose, the specification of the transfer points to a non-existing resource as the destination for the data. The transfer will fail. A tool can be used to intercept the failure notification and present the user with a visual message. The user can then proceed to modify the bogus specification, after which, the particular failing element can be restarted using the state present in the failure event.

8.1 Installation

8.1.1 Obtaining the Source Code

Karajan can be downloaded from the Java CoG Kit CVS archive. Instructions regarding the Java CoG Kit requirements and details on obtaining the Java CoG Kit sources, are available in Section 4.

8.1.2 Compiling Karajan

Change directory to *cog*/modules/karajan and type 'ant dist'. This will compile Karajan and all its dependencies. It will also create a *dist* directory containing the distribution of Karajan. Inside the *dist* directory, the *bin* directory will contain the necessary scripts that can be used to launch Karajan.

8.2 Using Karajan

There are two interfaces to Karajan:

1. The command line interface, accessible through *bin/karajan* provides a very simple interface, which is mainly non-interactive and does not provide feedback on the execution of the workflow.
2. The graphical interface, which can be started through *bin/karajan-gui*, can display a graphical representation of the workflow and other progress information and statistics. It also allows interaction with the workflow.

8.2.1 Command Line Interface

The command line interface allows you to start a workflow. The syntax is very simple:

```
>./karajan workflow.xml
```

Karajan will then try to load, parse, and execute the specified workflow. Any resulting messages will be printed on the console.

8.2.2 Graphical Interface

The graphical interface allows for additional interaction with the execution engine. It can be started using *bin/karajan-gui*. The following command line options are supported:

-help : Displays a brief usage summary

-load filename : Can be used to load a workflow upon starting

-run : Used in conjunction with *-load*, will immediately start the execution of the specified workflow.

When started without any parameters, an empty view is presented (Figure 8.1).

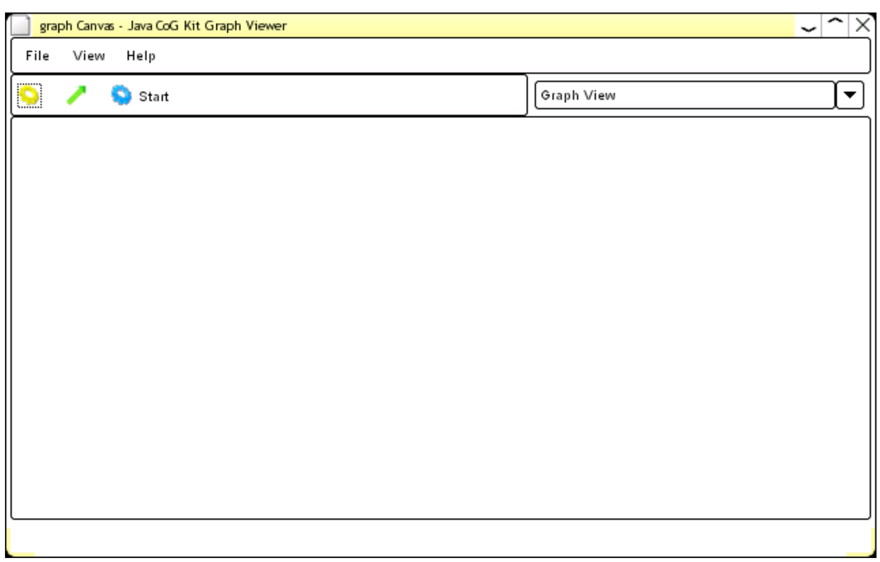


Figure 8.1: An Empty Karajan Desktop

The File->Open menu item can be used to load a workflow. After the workflow is loaded, a graph that represents the control flow of the loaded specification will be drawn. An example can be seen in Figure 8.2.

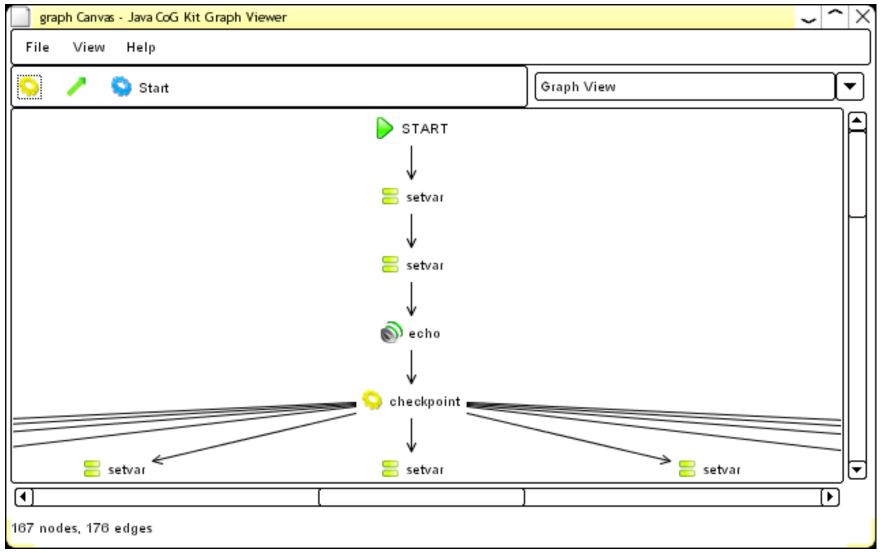
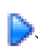


Figure 8.2: A Workflow Was Loaded

The workflow can be started by pressing the *Start* button located on the toolbar. Once started, the status of each node will be visible as an overlayed image over the node icon. The following states exist:

None : The node has not yet been executed

Running () : The node is being executed

Completed () : The node completed execution successfully

Failed () : Execution of the node failed

Breakpoint () : A breakpoint was set on the node

Paused () : The execution was paused at the current node, possible due to a breakpoint being set on the node

Setting A Breakpoint

Breakpoints can be set using a node's context menu. Clicking on a node with the right mouse button, will pop-up the menu, as it can be seen in Figure 8.3

Whenever the execution of the workflow reaches the node where a breakpoint was set, a message dialog will pop up, and the execution of the specific thread/branch where the node is located will be suspended (see Figure 8.4).

The execution can then be resumed using the context menu of the node (accessible by right-clicking on the node). In the case of a paused node, an item that will resume the execution will be present in the menu (see Figure 8.5).

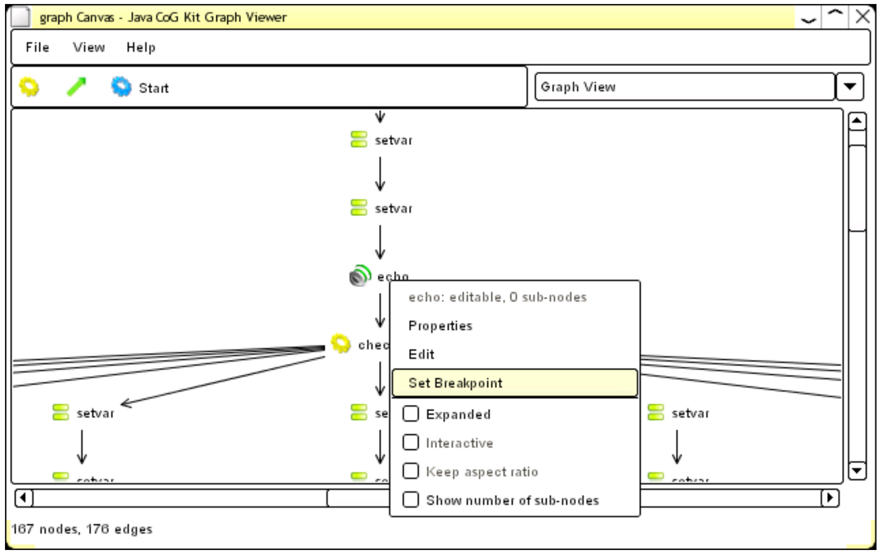


Figure 8.3: The Node Context Menu

Error Handling

Errors that may occur during the execution of the workflow, which are not explicitly handled in the workflow specification, will result in a dialog window that provides several options for dealing with the error. A sample error dialog is presented in Figure 8.6.

A description of each option provided by the error dialog is shown below:

Abort : Passes the error to the workflow engine, which will result in an error message dump on the console and the immediate termination of the workflow.

Ignore : Completely ignores the error as if it has never occurred.

Restart : Restarts the failed node. You can also specify the number of times that the node will be restarted before the execution is aborted.

Apply to all errors of this type : Whenever an identical error occurs on any node, the same action will be applied automatically.

Apply to all errors for this element : All other errors that occur on the node will automatically be treated with the same action.

8.3 Language Specification

The Karajan specifications are written in an XML based language. Extensive information about XML is available from `http://www.w3.org/XML`.

8.3.1 Concepts

Elements

The building block of a Karajan workflow is an XML element. The structure of Karajan workflows is very similar to that of structured languages (such as C, Java,

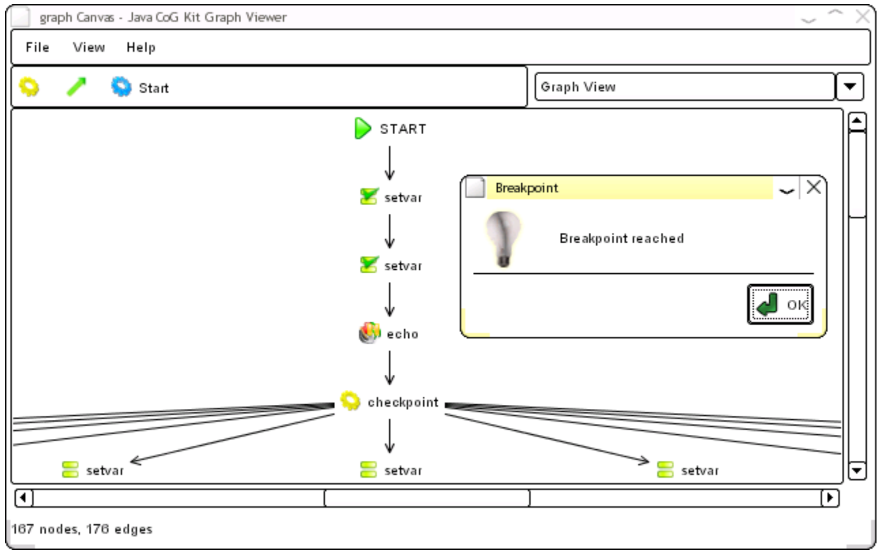


Figure 8.4: A Breakpoint Was Reached

Pascal, etc). Most elements can also act as containers for other elements. Each element performs a specific function, or describes how contained elements relate to each other.

Generally, Karajan imposes little restrictions on where particular elements can appear. However, a few elements can only appear inside other elements, but this restriction does not imply anything about the depth of the containment. The following example illustrates the previous sentence.

Given the elements **main** and **sub**, and given that **sub** can only appear inside **main**, both the following examples are valid:

1.
```
 ...
<main>
    ...
    <sub>
        ...
    </sub>
    ...
</main>
 ...
```

2.
```
 ...
<main>
    ...
    <other>
    ...
        <sub>
            ...
        </sub>
        ...
    </other>
    ...
</main>
```

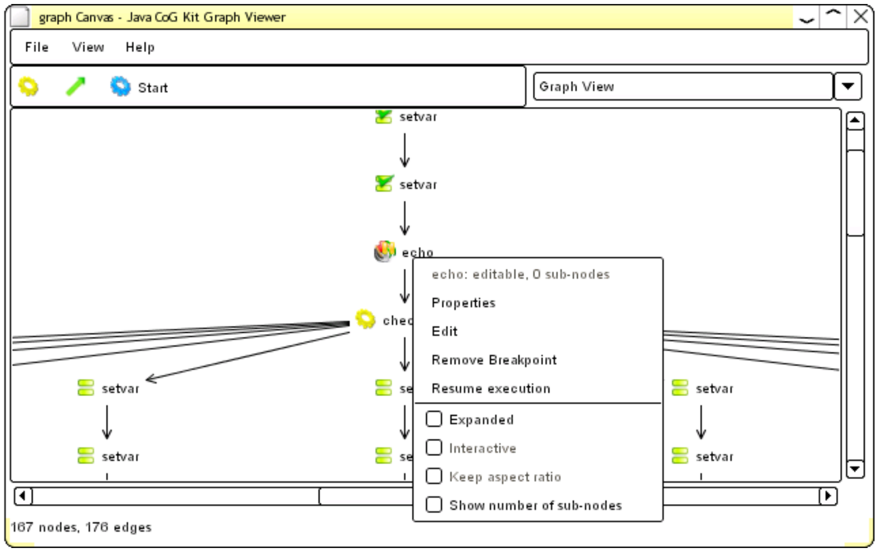


Figure 8.5: Resuming Execution

```
...
```

To eliminate ambiguity in such cases, the **sub** element will be denoted by **main**>**sub**.

Variables

Variables can be used in Karajan to store temporary values, values that can change and appear often in the specification, as counters for iterators, etc.

Defining Variables Variables can be defined explicitly using the **setvar** (8.8.33) element, which takes two attributes: *name* and *value*. The following example assigns the value *blah* to the variable named *variable1*:

```
<setvar name="variable1" value="blah"/>
```

If the *value* attribute is not specified, **setvar** (8.8.33) will use the value of the default return variable ($). This can be used for getting values from functions[1]:

```
<setvar name="variable2">
    <!-- read the contents of /tmp/exitcode             -->
    <function:readFile name="/tmp/exitcode"/>
</setvar>
```

Variable Expansion Variables can be expanded inside element attributes by enclosing them inside curly brackets. Nested expansion is also possible, but must be used with care.

Examples:

[1] more about functions in Section 8.3.8

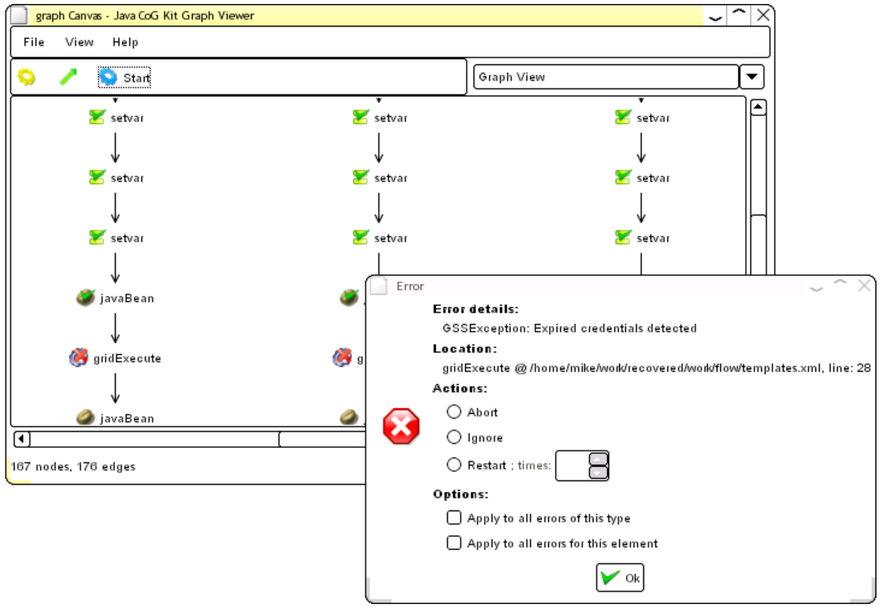


Figure 8.6: Error Dialog

1.
```
<setvar name="variable1" value="blah"/>
<echo message="variable1={variable1}"/>
```

2.
```
<!-- variable2 is not defined; the value will not be -->
<!-- expanded                                          -->
<setvar name="variable1" value="{variable2}"/>

<setvar name="variable2" value="blah"/>

<!-- at this point, both variable1 and variable 2      -->
<!-- are defined                                       -->
<!-- the first expansion will evaluate variable1 to    -->
<!-- {variable2}                                       -->
<!-- the second expansion will evaluate variable2 to   -->
<!-- "blah"                                            -->
<echo message="{variable1}"/>
```

3.
```
<setvar name="variable2" value="blah"/>

<!-- variable2 is now defined                          -->
<!-- variable1 will be assigned the value of "blah"    -->
<!-- directly                                          -->
<setvar name="variable1" value="{variable2}"/>

<echo message="{variable1}"/>
```

All three examples will print the same value: `blah`

The Scope of Variables The scope of variables is limited to the element inside which they appear, unless they are overridden in sub-elements. In such a case, the scope of the override will be limited to the element in which the variable was overridden. The following example illustrates this:

```
<sequential>
    <!-- define the variable "var"                              -->
    <setvar name="var" value="one"/>

    <!-- print its value on the console                         -->
    <echo message="{var}"/>

    <!-- a container                                            -->
    <sequential>

        <!-- override "var"                                    -->
        <setvar name="var" value="two"/>

        <!-- print the value on the console                     -->
        <echo message="{var}"/>

    </sequential>

    <!-- at this point "var" will be "one" again                -->
    <echo message="{var}"/>

</sequential>

<!-- "var" does not exist here                                  -->
<echo message="{var}"/>
```

The example will produce the following output:

```
one
two
one
{var}
```

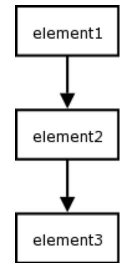
In the last **echo** (8.8.4) element, Karajan tries to expand *var*, but since it cannot be found, it prints the message literally.

8.3.2 Parallelism

Karajan supports two basic containers through which parallelism can be achieved, namely **sequential** (8.8.32) and **parallel** (8.8.22) . Both containers are synchronous, which means that their execution will terminate when all sub-elements have finished execution. This behavior can be overridden in any element, by specifying the *sync="false"* attribute. The following examples illustrate the use of **sequential** (8.8.32) and **parallel** (8.8.22) containers, as well as synchronous and asynchronous execution. On the right side, an image showing the resulting control flow of the specifications on the left is shown:

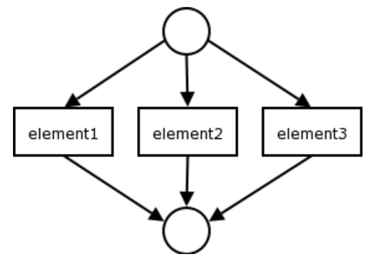
Sequential execution

```
<sequential>
    <element1/>
    <element2/>
    <element3/>
</sequential>
```



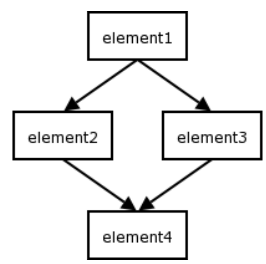
Parallel execution

```
<parallel>
    <element1/>
    <element2/>
    <element3/>
</parallel>
```



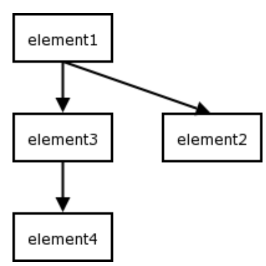
Mixed sequential/parallel execution

```
<sequential>
    <element1/>
    <parallel>
        <element2/>
        <element3/>
    </parallel>
    <element4/>
</sequential>
```



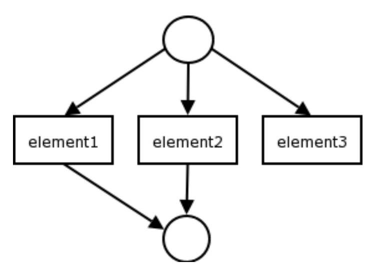
Sequential execution with asynchronous element

```
<sequential>
    <element1/>
    <element2 sync="false"/>
    <element3/>
    <element4/>
</sequential>
```



Parallel execution with asynchronous element

```
<parallel>
    <element1/>
    <element2/>
    <element3 sync="false"/>
</parallel>
```



8.3.3 Iterators

Iterators are used in Karajan to execute a part of the workflow repetitively. All iterators can have either a sequential behavior (the default), in which an iteration begins execution only after the previous iteration has completed execution, or a parallel behavior (switched on using the *parallel="true"* attribute), in which all iterations are executed in parallel. The parallel behavior of an iterator does not apply to contained elements. If a parallel iterator has two sub-elements, the elements will execute in sequential order. This can be prevented using an explicit **parallel** (8.8.22) container inside the iterator.

Two iterators exist: **for** (8.8.7) and **foreach** (8.8.8) . Both have a mandatory attribute (*name*) which represents the name of the iteration variable.

- The **for** (8.8.7) iterator is used for iterations across integer ranges and takes two numerical attributes, representing the first and last value. There are two equivalent ways of specifying the two attributes, as shown in the following examples:

```
<for name="iteration" from="1" to="4">
    <echo message="Iteration {iteration}"/>
</for>
```

  and

```
<for name="iteration" range="1,4">
    <echo message="Iteration {iteration}"/>
</for>
```

  The result in both cases is:

```
1
2
3
4
```

- The **foreach** (8.8.8) iterator can be used to iterate across arbitrary values, or files in a directory (a feature that needs a little polishing). Iteration values can be specified using the *in* attribute. To iterate on the files in a directory, the *dir* attribute can be used.

  Examples:

```
<foreach name="iteration" in="one, two, three, four">
    <echo message="Iteration {iteration}"/>
</for>
```

  would produce the following output:

```
one
two
three
four
```

  while

```
<for name="file" dir="/home/johndoe">
    <echo message="File {file}"/>
</for>
```

  would list all the files contained in the home directory of user *johndoe*.

8.3.4 Templates

Templates can be used to define reusable code, and are somewhat similar to procedures in other languages. Templates can accept named parameters.

Definition of templates can be done using the **templateDef** (8.8.35) element. The mandatory attribute *name* specifies the name of the template. The body of the template can consist of any Karajan elements. An additional element (**default** (8.8.3)) can be used to designate default values for parameters. A simple template definition is shown below:

```
<templatedef name="sample">
    <default name="arg1" value="default1"/>
    <default name="arg2" value="default2"/>
    <echo message="arg1 is {arg1}"/>
    <echo message="arg2 is {arg2}"/>
    <echo message="arg3 is {arg3}"/>
</templatedef>
```

Template invocations can be made via the **template** (8.8.34) element, which accepts the *name* attribute, plus any number of other arguments that are passed to the template:

```
<template name="sample"
    arg1="value1"
    arg2="value2"
    arg3="value3"/>
```

All templates are re-entrant as long as no external resources are involved. Variables defined or overridden inside templates are considered local.

8.3.5 Grid-related Elements

Karajan contains a series of elements that are divided into three main categories: grid resource description, grid tasks, and configuration.

Grid Resource Description Elements

- **scheduler>grid** (8.8.26) encapsulates a set of resources that will be used by the scheduler. Accepts an optional *name* attribute.
- **scheduler>grid>host** (8.8.27) designates a single contact point (a remote host). The mandatory *name* attribute denotes the hostname of the remote contact. A *cpus* attribute allows the specification of the number of CPUs the host has. This information may be used for scheduling purposes. The *host* variable is available inside the **scheduler>grid>host** (8.8.27) element, and has the value of the *name* attribute.
- **scheduler>grid>host>service** (8.8.28) defines a host service. The *version* attribute allows the definition of a logical handle that can be used to group multiple services based on technology/version. The *type* attribute specifies the service type. The current possible values are *job-submission*, and *file-transfer*. The exact details of the service are expressed in the form of a URL. The format and details of the service URLs differ from handler to handler.

The Handlers Section 8.4 provides details of all supported handlers and their details.

The following example illustrates the use of the above elements:

```
<grid name="default">
    <for name="index" from="1" to="20">
        <host name="lg0n{index}.pts.uml.mov" cpus="1">
            <service
                version="gt2-2.4.0"
                type="job-submission"
                url="{host}:2119/jobmanager-fork"/>
            <service
                version="gt2-2.4.0"
                type="file-transfer"
                url="gsiftp://{host}:2811"/>
        </host>
    </for>
</grid>
```

Grid Tasks

Two types of grid tasks are available: remote execution and transfer. For all tasks that require one or more host attributes, the hosts may need to exist in the grid definition, such that the handlers can extract service contact information. In some cases, defaults may work.

- **gridExecute** (8.8.14) can be used to submit a remote job. The attributes are[2]:

host* : Specifies the host to which the job will be submitted. The host must exist in the grid description, such that the handler can extract the correct service information. If the attribute is not specified, it is up to the scheduler to pick a host for this job.

executable : The executable to be run. It must exist on the remote site. If it does not, it can be transfered beforehand using a transfer task.

args* : The attributes to be passed to the executable.

stdin* : If input redirection is desired, this attribute can be used to specify a remote file that will be redirected to the process' standard input.

stdout* : Can be used to redirect the standard output of the job to a remote file.

stderr* : Used optionally to redirect the standard error stream of the job to a remote file.

- **gridTransfer** (8.8.15) is used to transfer a file from one host to another. The accepted attributes are:

srchost : The source host. Use *localhost* for the local machine.

srcdir : The source directory where the file can be found.

srcfile : The name of the file that is to be transfered.

desthost : The destination host. Can also be *localhost* for the local machine.

[2] attributes followed by an asterisk are optional

destdir : The directory on the destination host where the file will be placed.

destfile* : Can be used to rename the file during the transfer.

- It may sometimes be necessary to execute a set of tasks on the same host. The **allocateHost** (8.8.1) element can be used for this purpose. The *name* attribute specifies a variable that can be used inside the element by the various tasks whenever the remote host needs to be referenced. A simple example is provided below:

```
<allocateHost name="remote">
    <!-- transfer the input data                    -->
    <gridTransfer
        srchost="localhost"
        srcdir="/tmp"
        srcfile="in"
        desthost="{remote}"
        destdir="/tmp"/>

    <!-- do the heavy processing                    -->
    <gridExecute
        host="{remote}"
        executable="/usr/bin/tac"
        args=""
        stdin="/tmp/in"
        stdout="/tmp/out"/>

    <!-- transfer back the results                  -->
    <gridTransfer
        srchost="{remote}"
        srcdir="/tmp"
        srcfile="out"
        desthost="localhost"
        destdir="/tmp"/>
</allocateHost>
```

Configuration Elements

The configuration elements are used to configure the scheduler and the handlers. Following is a list of grid-related configuration elements:

- **scheduler** (8.8.25) is used to select a scheduler type and specify various parameters for it. Currently only one scheduler is available (named *default*). The attributes are:

type : The type of the scheduler desired. Only *default* is available at this time.

jobsPerCpu : Sets the maximum number of tasks that the scheduler will allocate for one CPU.

maxSimultaneousJobs : Sets the total maximum number of remote tasks that the scheduler will allow at any given time.

showTaskList : If set to *true* the scheduler will pop-up a window providing a lists of tasks that are being executed, and additional task and memory statistics.

- **scheduler>taskHandler** (8.8.29) selects the type of Java CoG Kit Core task handler that is going to be used by the scheduler. The attributes are *type*, which selects the type of the handler (for a list of supported handlers, consult the Supported Handlers Section 8.4), and *version*, which is used by the scheduler to select the appropriate grid resources by matching it with the *version* attribute in the **scheduler>grid>host>service** (8.8.28) elements. Multiple handlers can be specified. In this case, all of them will be used, the highest priority going to the one listed first.
- **scheduler>taskHandler>securityContext** (8.8.30) is used to define a security context (passwords, private/public keys, proxy location, etc.) for the task handler. The mandatory attribute is the *name* attribute. A series of sub-elements can exist for the **scheduler>taskHandler>securityContext** (8.8.30) element. These sub-elements are handler specific and are described in Section 8.4.

8.3.6 Explicit Error Handling

In certain cases, errors that appear in certain locations, are known to have no impact on the overall execution of a workflow. A typical example would be a cleanup process. In such cases, it may be preferable to be able to simply ignore errors. Other operations have particularly high rates of failure. However, subsequent re-executions of such operations may result in a successful result. The following elements deal with such cases:

- **ignoreErrors** (8.8.16) has no attributes and any errors that occur on contained elements are ignored.
- **restartOnError** (8.8.24) has a numeric mandatory attribute (*times*) that specifies the number of times the contained sub-workflow is restarted when an error occurs, before that error is reported.
- **generateError** (8.8.13) will cause an error to be generated, with an associated message specified by the *message* attribute.

8.3.7 Miscellaneous Elements

- **project** (8.8.23) is the main container of a workflow. Any workflow specification that can be executed by Karajan must have **project** (8.8.23) as the root element.
- **echo** (8.8.4) echoes a message on the console. The message can either be specified using the *message* attribute, or by inlining the text inside the **echo** (8.8.4) element.
- **include** (8.8.17) can be used to include re-usable parts of workflows inside a workflow specification. The **include** (8.8.17) element has no function for the workflow. It acts during the parsing process, and before the actual execution begins. The *file* attribute specifies a file, relative to the main specification file, that will be substituted for the **include** (8.8.17) element. The included file will have its root element ignored. Section 8.5 provides details about the include search path.
- **executeJava** (8.8.6) will synchronously execute a Java program. The *mainClass* attribute can be used to specify the fully qualified class name that

contains the main method. The class must be present in the classpath of the instance of the Java Virtual Machine that is executing the workflow.

- **javaBean** (8.8.19) can be used to set properties and invoke methods on a JavaBean. The fully qualified class name of the JavaBean must be specified using the *className* attribute. The following sub-elements are available inside a **javaBean** (8.8.19) element:

  – **javaBean>setProperty** (8.8.21) will set a property on an instance of a JavaBean. The *name* attribute indicates the name of the property that is to be set. Karajan will then look for a setter method corresponding to the given name. The *type* attribute specifies the class type of the value. The supported types are *String*, *Integer*, *Float*, *Double*, and *Boolean*. The value for the property is given by the *value* attribute.

  – **javaBean>invokeVoid** (8.8.20) invokes a method on the JavaBean which has no arguments. The name of the method is given by the *methodName* attribute.

- **wait** (8.8.36) produces a delay in the execution. One of the *delay* or *until* attributes must be set. The *delay* attribute indicates wait period in milliseconds, while the *until* attribute specifies an absolute date in a format accepted by the `java.util.Date` class.

8.3.8 Functions

Relatively few functions are defined in Karajan. We hope to provide a more complete set in the future. All functions put their return value in the default return variable (*$*). The defined functions are:

- **function:contains** (8.8.9) determines whether a file contains a specific sequence of characters. The *file* attribute points to the file to be checked, while the *value* attribute specifies the value to be searched.

- **function:numberFormat** (8.8.10) allows the formatting of a decimal number. The *pattern* attribute indicates the patter to be used for formatting (as used by the `java.text.DecimalFormat` class). The *value* attribute holds the decimal value that is to be formatted.

- **function:readFile** (8.8.11) reads the contents of a file, pointed to by the *name* attribute. This is intended for short text files that may possibly hold things like error messages or exit codes. The file is completely read into memory, therefore this function would not be suitable for manipulation of large files.

- **function:UID** (8.8.12) generates a unique ID[3].

8.4 Supported Handlers

Karajan supports any handler that the Java CoG Kit Core supports. However, some handlers may require particular security settings, which must be known, or the handler will not work. Karajan can pass such settings to Core, using generic attributes. The following are default Core handlers, together with examples that show their usage in Karajan.

[3] This function is not thread-safe at the moment of this writing, but plans are to correct the problem

GT2 : This handler does not require any specific settings, but the Java CoG Kit must be configured properly for the handler to work.

```
<!-- define the task handler to be used -->
<taskHandler type="GT2" version="gt2-2.4.0">

    <!-- associate with the previously defined -->
    <!-- security context -->
    <securityContext name="gt2"/>

</taskHandler>

<!-- define a small grid -->
<grid name="default">
    <host name="cold.mcs.anl.gov" cpus="2">
        <service
            version="gt2-2.4.0"
            type="job-submission"
            url="{host}:2119/jobmanager-fork"/>
        <service
            version="gt2-2.4.0"
            type="file-transfer"
            url="gsiftp://{host}:2811"/>
    </host>
</grid>
```

GT3 : Similar to the GT2 handler, the GT3 handler does not require any special parameters. The following example shows how the GT3 handler can be used in Karajan:

```
<!-- define the task handler to be used -->
<taskHandler type="GT3" version="gt3-3.0.2">

    <!-- associate with the previously defined -->
    <!-- security context -->
    <securityContext name="gt3"/>

</taskHandler>

<!-- define a small grid -->
<grid name="default">
    <host name="mild.mcs.anl.gov" cpus="2">
        <service
            version="gt3-3.0.2"
            type="job-submission"
            url="http://{host}:8080/ogsa/services/base/
            gram/MasterForkManagedJobFactoryService"/>
        <service
            version="gt3-3.0.2"
            type="file-transfer"
            url="http://{host}:8080/ogsa/services/base/
            multirft/MultiFileRFTFactoryService"/>
    </host>
</grid>
```

SSH : The SSH handler requires explicit pointers to the credentials used for authentication. It supports both username/password (which we do not recommend) and public key authentication. The following example shows how to use an SSH Core handler with Karajan:

```
<!-- define the task handler to be used -->
<taskHandler type="SSH" version="ssh">

    <!-- associate with the previously defined -->
    <!-- security context -->
    <securityContex name="ssh-doe">
        <property
            name="ssh-username"
            value="johndoe"/>
        <property
            name="ssh-private-key"
            value="/home/johndoe/.ssh/identity"/>
        <property
            name="ssh-pass-phrase"
            value="guessme"/>

    <!-- "ssh-password" could also be used instead of-->
    <!-- the ssh-private-key/ssh-pass-phrase pair -->
    </securityContext>

</taskHandler>

<!-- define a small grid -->
<grid name="default">
    <host name="hot.mcs.anl.gov" cpus="2">
        <service
            version="ssh"
            type="job-submission"
            url="{host}:22"/>
        <service
            version="ssh"
            type="file-transfer"
            url="{host}:22"/>
    </host>
</grid>
```

8.5 Include Search Path

When the **include** (8.8.17) element is used, the specified file is first searched in the directory where the main workflow file is located. If the requested file is not found, the include search path is iterated until the file is found. The include search path is defined in *etc/karajan.properties*. The list of directories is separated by colons. A special token, *@classpath*, indicates Karajan should try to find the file in the JVM class path.

8.5.1 System Defaults

By default, Karajan starts with a very bare set of elements defined. In order to access most of the above elements, you should include the *sysdefaults.xml* file in the beginning of your workflow:

```
<project name="myproject">
    <include file="sysdefaults.xml"/>
    ...
</project>
```

8.6 Architecture

This section explains the main architectural characteristics of Karajan.

8.6.1 The Loading Process

Karajan workflows are specified using an XML syntax. Only basic structural and syntactic validation is being performed at load time. Semantic validation is performed individually at execution time by each execution element.

A one-to-one mapping of the XML document elements and flow elements is done using an element map which provides the correspondence between XML element names and fully qualified flow element class names. An exception applies to the **include** (8.8.17) element, which immediately after being loaded instructs the loader to parse the included file, the contents of which is in-lined in the current element tree.

8.6.2 The execution model

There are two important notions to remember in Karajan. One is the execution element (or flow element), which (as outlined in the previous subsection) is constructed from XML elements in the specification. The second one is the event. Events are used either to notify elements about the status of the execution of other elements, or to instruct elements to perform certain actions (such as start or restart execution). Events also encapsulate the state of the workflow through a variable stack. The stack contains the complete run-time state of the workflow for a specific thread of execution. There should be no deterministic difference in the execution of two different instances of the same type of element, with the same attribute values, that receive equivalent events (with identical states).

Elements are static for the workflow. Their internal state may change during the execution of the workflow, but the workflow state must not be influenced by the internal state of the elements. They react to events and can use the stack passed through the events to manipulate the state of the workflow. Each element that is being executed can add a frame to the variable stack. The frame can be used to store variables that can represent the state of the element. These variables are also accessible to contained elements. When the execution of an element ends, it destroys the frame that it created, and together with it the variables that it contained. This behavior is not enforced, but it is recommended.

In the case of parallel containers, each parallel thread will start with a copy of the stack. The stack copies will internally share frames that are not write accessible to the threads. A diagram detailing the stack model can be seen in Figure 8.7. The conventional representation *pop - (a b – a)* for a stack indicates that *a* and *b* were present on the stack before the execution of *pop*, and only *a* was left afterwards.

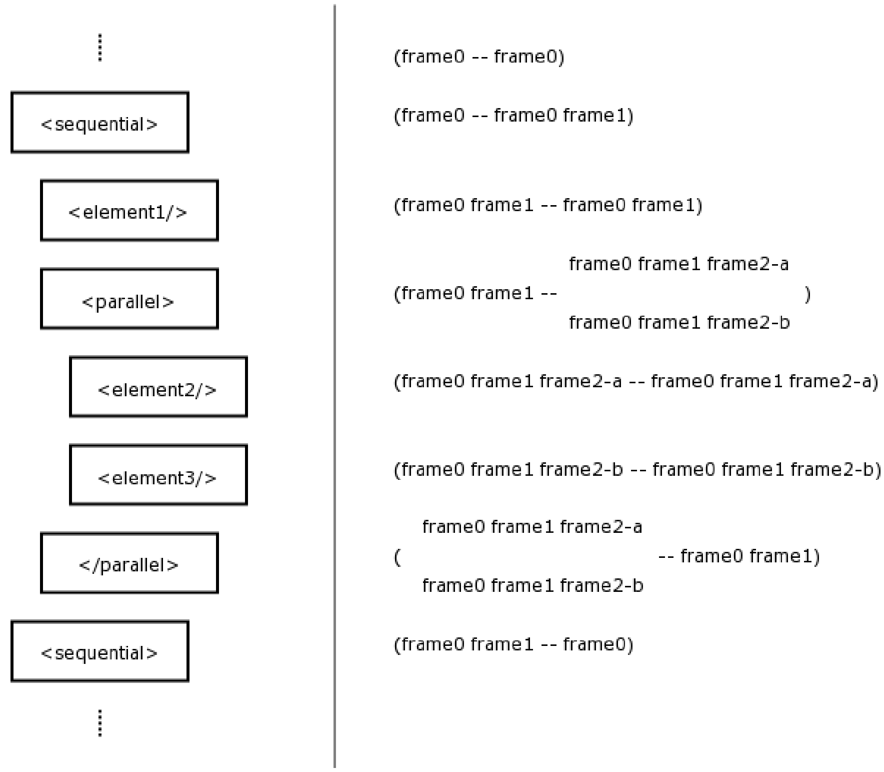


Figure 8.7: The Stack Model

The above may in some cases be insufficient. Certain variables need to be made accessible to the parent frames such that they can be used by subsequent elements that are not descendants of the element which defined those variables. This is still only possible in a sequential context. There is no way to propagate information from one thread to the other. While this characterizes the workflow execution, the applications themselves can still use inter-thread or inter-process communication/messaging as needed.

A distinction exists between Karajan threads and Java or OS threads. The Karajan threads differ between each other only by the variable stacks they receive. No assumption can be made about the Java or OS thread in which a Karajan thread executes. The events that are passed between elements are managed by an event dispatcher (which may use more than one Java thread). The appearance of parallelism is achieved through the fact that elements either take a short time to execute or they make use of their own Java threads if known to take a longer time to execute. The result is the ability to execute a large number of Karajan threads, without the overhead required by Java/OS threads. As an example, each Java thread requires a minimum of 96 Kilobytes of memory just for the thread stack.

Karajan defines six event types:

START : tells an element it should start execution.

RESTART : tells an element which has not completed execution yet, that it should restart its execution.

EXECUTION_STARTED : sent by an element immediately after it has started execution.

EXECUTION_COMPLETED : sent by an element after it completes execution. This event is sent as a result of receiving the END element and after cleanup is done.

EXECUTION_FAILED : generated by an element when the execution failed. The frame created by the element should be popped from the stack before the event is sent.

EXECUTION_RESTARTED : generated after receiving a restart event.

An example of the execution model for both a sequential and a parallel container can be seen in Figure 8.8, and Figure 8.9 respectively.[4]

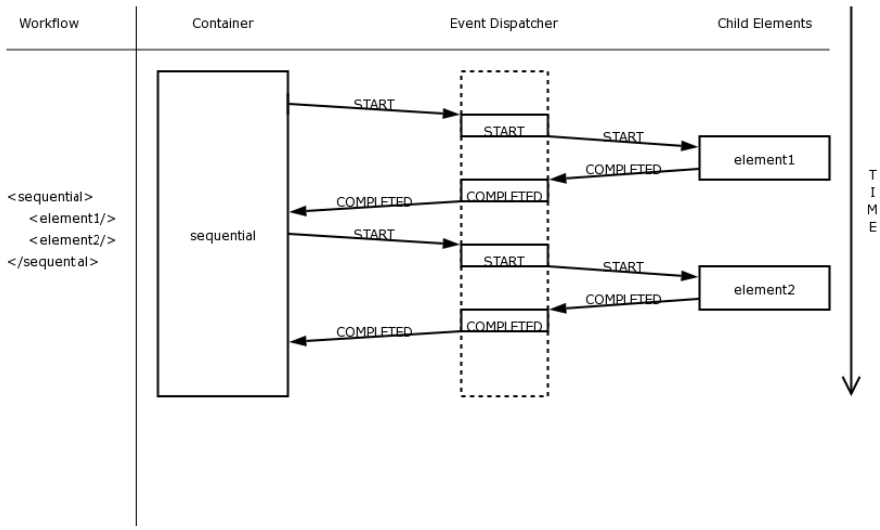


Figure 8.8: Execution of Sequential Elements

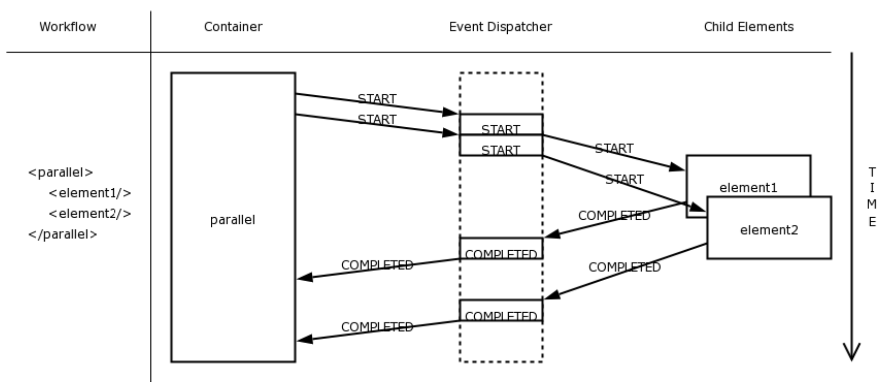


Figure 8.9: Execution of Parallel Elements

8.6.3 Task Scheduling

Task scheduling on grid resources is done using a scheduler that in turn uses the Java CoG Kit Core Grid abstraction layer. The **gridExecute** (8.8.14) and **grid-**

[4] For space and readability considerations, the EXECUTION_COMPLETED event type was shortened to COMPLETED in the images

Transfer (8.8.15) elements submit the requests for execution to the scheduler which enqueues these requests and executes them as resources become available. It is up the the scheduler to manage both local and remote resources in order to ensure that these resources are not overloaded. However, certain parameters can be passed to the scheduler (using the **scheduler** (8.8.25) element) that can alter the way in which the resources are allocated from the defined pool (the **scheduler**>**grid** (8.8.26) element). It is also the duty of the scheduler to choose the proper handlers and services for a given task.

Tasks may or may not have certain constraints associated with them. Some tasks may have pre-defined resources or handlers that they require. For example a certain job submission may have a predefined resource that it needs to run on. In such a case, the scheduler should not attempt to find another resource for the task. However, when a task does not specify such constraints, the scheduler must fill the missing parts required to execute the task. The scheduler must also take care of task encapsulation. This refers to the case when certain tasks must be executed on the same resource.

When trying to submit a task, the default scheduler cycles through the list of available resources and uses the first one that it finds suitable for the given task. The resource search for the next task begins with the resource immediately following the last used resource in the list. If the end of the list is reached, the search continues from the beginning of the list. If after one complete cycle through all the elements in the list, nothing suitable is found, the execution is postponed for a later time, when some of the resources may become free.

The scheduler does not take care of dependencies between tasks or the order of the execution of tasks. It is up to the workflow engine to do so. For the default scheduler, once a set of tasks is queued in the scheduler, there is no way to know anything about the order in which the execution of these tasks will begin, nor about the order in which they will complete their execution. Of course, other schedulers for which such things are known can be written, but the scheduler interface does not define explicit ways for enforcing execution order, nor it is required by the workflow engine from the scheduler that such order be known or be specifiable.

8.7 Checkpointing

Checkpointing is still an experimental feature in Karajan. This Section describes the basic workings of checkpointing in Karajan. Checkpointing parameters can be adjusted using the **checkpoint** (8.8.2) element. Checkpointing here refers to workflow checkpointing. Only the state of the workflow is saved in a checkpoint. The application state is not included in the checkpoint. Imagine the following scenario (in chronological order): a workflow creates certain files on a remote resource, the workflow is checkpointed and interrupted, then the files are deleted, and the workflow is restored from the checkpoint. If the files are further needed and referenced in the workflow, an error will eventually occur.

8.7.1 Checkpoint Creation

Checkpointing works by dumping the workflow definition and workflow state to a file. The workflow definition consists of the element tree, and is similar to the workflow source after all the **include** (8.8.17) elements have been processed. The state of the workflow is composed of two main areas: The set of events that are waiting to be delivered, and the state of elements that have begun execution but have not yet completed it.

When a checkpoint is requested, the checkpoint manager first locks the event dispatcher in order to guarantee that the state of the workflow remains consistent during the checkpointing process. While the event dispatcher is locked, it does not deliver events, nor it accepts new events. Threads that are trying to post events to the dispatcher are suspended during this time. The event dispatcher also keeps track of elements that have been started but were not completed and also keeps a reference to the stack of those elements. Since the event dispatcher does not make a full copy of the element stack (for performance considerations), it may sometimes be the case that an element can at specific moments modify the stack and leave it in an inconsistent state. A special locking mechanism that allows an element to group operations on the stack that should be atomic is provided. The checkpoint manager will therefore wait until all elements have completed the execution of blocks that need to be atomic relative to the stack, before making the actual checkpoint. It can be easily seen that posting an event to the dispatcher inside a atomic block could cause a deadlock. It is thus prohibited to do so.

After all the checkpoint manager has ensured that the overall state of the workflow is in a consistent state, it begins writing the specification, events and list of currently executing elements to the checkpoint file. Each event and running element has an associated stack, which will also be serialized. It is mandatory that all elements put only Java Beans on the stack, otherwise variables on the stack will not be saved, leading to an incomplete checkpoint.

8.7.2 Restoring from a Checkpoint

When invoked with a checkpoint file from the command line, Karajan will automatically detect the checkpoint and restore the state of the workflow at the time the checkpoint was taken. A checkpoint file is self contained, and does not require the original workflow description.

The restoration process is done by first loading the workflow specification from the checkpoint file. Afterwards, pending events are deserialized and posted to the event dispatcher. Elements that were executing at the time the checkpoint was taken are also sent a RESTART event using the associated stack that was saved during checkpointing. This will effectively put the workflow in the state it was at the time the checkpointing was done.

8.8 Quick Element Reference

8.8.1 allocateHost

Defines a token that can be used to guarantee that a set of tasks will be executed on the same resource.

Attributes:

name: The name of the variable that should be set with the value of the token. The token can then be used by the **gridExecute** (8.8.14) and **gridTransfer** (8.8.15) elements as a host attribute.

8.8.2 checkpoint

Sets checkpointing parameters or forces the immediate creation of a checkpoint.

Attributes:

fileName: The name of the file to which the checkpoint will be written.

interval: Sets the interval at which regular checkpoints will be performed. The interval is specified in seconds.

now: If set to *true* causes the immediate creation of a checkpoint. This is merely a debugging feature. The recommended method is to set a regular interval for checkpointing.

8.8.3 default

Typically used to define the default value for an argument in a template. It sets the value of the specified variable if it is not already defined.

Attributes:

name: The name of the variable to be defined

value: The value of the variable

8.8.4 echo

Echoes a message on the console

Attributes:

message: The message to be echoed

<inline text>: Can be used instead of the *message* attribute for larger chunks of text

8.8.5 elementDef

Defines a new workflow element

Attributes:

className: The fully qualified Java class name of the element

nodeType: The XML element name to be defined

8.8.6 executeJava

Executes a Java application in a separate thread. The element completes execution when the application completes execution.

Attributes:

mainClass: The fully qualified name of the class that contains the main method.

8.8.7 for

Iterates across a range of integer values

Attributes:

from: Used in conjunction with the *to* attribute indicates the first value of the iteration

name: The name of the variable that is set with the current iteration value

parallel: If set to *true* the iterations will be executed in parallel, otherwise they will be executed sequentially

range: A range of the form *n, m* describing all integers between *n* and *m* (inclusive)

to: Used together with the *from* attribute, indicates the last value of the iteration

8.8.8 foreach

Iterates across a sequence of discrete values

Attributes:

dir: Points to a directory. The iteration will be performed using the files in the specified directory.

in: A comma separated list of strings that will be used as iteration values

parallel: If set to *true* the iterations will be executed in parallel, otherwise they will be executed sequentially

8.8.9 function:contains

Tests if a file contains a certain text.

Attributes:

file: The file to be searched

value: The value to be searched

8.8.10 function:numberFormat

Formats a number according to the specified pattern.

Attributes:

pattern: The pattern according to which the number is formatted. The pattern has the syntax used by `java.text.DecimalFormat`

value: The value to be formatted

See also: *http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html*

8.8.11 function:readFile

Reads a file and stores the contents into a variable. This function is intended for small files.

Attributes:

file: The file to be read

8.8.12 function:UID

Generates a unique numeric ID

8.8.13 generateError

Causes an error to be generated

Attributes:

message: Sets the message associated with the error

8.8.14 gridExecute

Executes a job on the grid.

Attributes:

args: The arguments to be passed to the executable

directory: The directory on the remote resource to execute the job in

executable: A path to an executable on the remote resource

host: A resource on which the job will be executed. If left empty, the scheduler will choose a resource. If a resource token (see **allocateHost** (8.8.1)) is used, the job will be executed on the resource that the token resolves to.

stderr: A path to a file on the remote resource to which the standard error stream of the executable is to be redirected

stdin: A path to a file on the remote resource that will be redirected to the standard input of the executable

stdout: A path to a file on the remote resource that will be used to redirect the standard output stream of the job.

8.8.15 gridTransfer

Used to transfer a file on the grid

Attributes:

destdir: The destination directory

destfile: The name of the destination file

desthost: The destination resource

srcdir: The source directory

srcfile: The source file

srchost: The source resource

8.8.16 ignoreErrors

Causes any errors generated by contained elements to be ignored

8.8.17 include

Parses the contents of a file inserting the elements after the position of the **include** (8.8.17) element.

Attributes:

file: The file to be included

8.8.18 nonCheckpointable

Has no functional purpose. It is generated inside serialized versions of events in the locations where non checkpointable elements are found. An example of such an element is **include** (8.8.17) which serves its purpose during the parsing process and has no further function afterwards.

8.8.19 javaBean

Allows instantiation and manipulation of Java Beans.

Attributes:

className: The fully qualified name of the bean class

See also: **javaBean>setProperty** (8.8.21) , **javaBean>invokeVoid** (8.8.20)

8.8.20 javaBean>invokeVoid

Invokes a method on the Java Bean that takes no arguments.

Attributes:

methodName: The name of the method to be invoked

See also: **javaBean** (8.8.19)

8.8.21 javaBean>setProperty

Sets a property on the Java Bean. It tries to do so by invoking the setter method for the property.

Attributes:

name: The name of the property to set

type: The type of the property to set. Supported types are: String, Integer, Float, Double, and Boolean

value: The value to set

See also: **javaBean** (8.8.19)

8.8.22 parallel

Executes the contained elements in parallel

8.8.23 project

The root container of a main workflow file.

Attributes:

name: The name of the project

8.8.24 restartOnError

Restarts the execution of the contained elements if an error is generated

Attributes:

times: Indicates the maximum number of times the contained elements should be restarted in case of an error before the error is reported.

8.8.25 scheduler

Specifies the type and parameters for the scheduler that is going to be used to schedule grid tasks.

Attributes:

type: Indicates the type of the scheduler. Details about available schedulers can be found in Subsection **??**

<*varies*>: Attributes to be passed to the scheduler.

8.8.26 scheduler>grid

Encapsulates a collection of grid resources that are used by the scheduler to schedule tasks.

8.8.27 scheduler>grid>host

Describes one resource in the grid definition

Attributes:

cpus: The number of CPUs that the resource has

name: The host name of the resource

8.8.28 scheduler>grid>host>service

Defines a service for a resource

Attributes:

type: The type of the service. Currently the accepted values are *job-submission* and *file-transfer*.

url: A URL indicating the location of the service.

version: A version label that is matched against the version labels of the defined task handler(s).

8.8.29 scheduler>taskHandler

Defines a task handler that can be used by the scheduler to execute tasks.

Attributes:

type: The type of the handler. Valid types are described in Section 8.4

version: A label used to match **scheduler>grid>host>service** (8.8.28) definitions against handlers

8.8.30 scheduler>taskHandler>securityContext

Used as a sub-element of **scheduler>taskHandler** (8.8.29) to define a security context for the handler.

Attributes:

type: Indicates the type of the security context. For details consult Section 8.4

8.8.31 scheduler>taskHandler>securityContext>property

Defines a property for a security context.

Attributes:

name: The name of the property

value: The value of the property

8.8.32 sequential

Executes the contained elements in sequential order

8.8.33 setvar

Sets the value of a variable

Attributes:

name: The name of the variable

value: The value of the variable

8.8.34 template

Invoked a template that was previously defined using **templateDef** (8.8.35)

Attributes:

name: The name of the template to be invoked

<varies>: Arguments to be passed to the template

8.8.35 templateDef

Defines a template

Attributes:

name: The name of the template to be defined

8.8.36 wait

Delays the execution for a period of time or until a specific time

Attributes:

delay: The delay in milliseconds to wait

until: A string representing a date. The format of the date is any format accepted by the `java.util.Date` class

See also: *http://java.sun.com/j2se/1.4.2/docs/api/java/util/Date.html*

9 Graph Editor

In order to compile the editor, execute 'ant dist' in the grapheditor directory. A new directory named 'dist' will be created. The 'dist' directory will contain the following subdirectories: bin - contains the launchers used to start programs from the command line. etc - contains configuration files (none at the moment) lib - contains the jar files needed to run the viewer/editor examples - a few graph examples and a perl client that can interact with the viewer service

9.1 Configuring

The 'dist/etc' directory contains the *grapheditor.properties* file, which can be used to customize certain aspects of the editor.

9.2 Running

To run the editor, cd to the 'dist/bin' directory and execute './grapheditor' (or grapheditor.bat on windows)

The following command line options can be used:

-s <port> : Starts the editor in server mode, listening for incoming connections on the specified port. If no port is specified, the default (9999) will be used.

-h | -help : Displays a list of options together with brief explanations.

-l | -load <file> : Loads <*file*> after starting up.

-t | -target <target> : Starts on the specified rendering target. The following targets exists as of the writing of this manual:

- swing 9.3.1 : Uses the Java Swing graphical interface. It is currently the only target that supports interactive editing.
- html 9.3.2 : Produces a HTML file together with any necessary images.
- postscript 9.3.3 : Renders the graph in an Encapsulated PostScript file.
- remote 9.3.4 : Can be used to forward the display of a graph to a remote viewer. The API can be used as if working with local rendering, and the remote renderers will take care of forwarding the events to the remote viewer. It is unlikely that this target would be of any use when invoking the graph editor from the command line.

Additional options can be specified in the `etc/grapheditor.properties` file. These options are generally particular to every target, and thus explained in the target descriptions subsections.

9.3 Using The Graph Editor

This section will describe how each target of the graph editor can be used.

9.3.1 The Swing Target

When started in the Swing (default) target, an empty frame (shown in Figure 9.3) is displayed. The three essential elements that can be identified are:

The Menu Bar : can be used for various operations like loading and saving of graphs. The menus are dynamic, which means that depending on certain factors (i.e. the selected view), their structure can change. The *View* menu can be used to select the active view. A snapshot showing the list of available views is shown in Figure 9.1

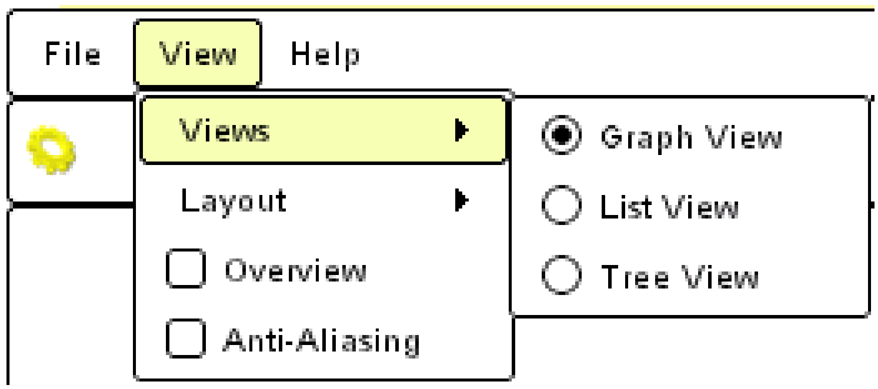


Figure 9.1: The Available Views

The graph view also supports a number of layout algorithms. These algorithms can be seen in the *View¿Layouts* menu, when the Graph View is selected (see Figure **??**).

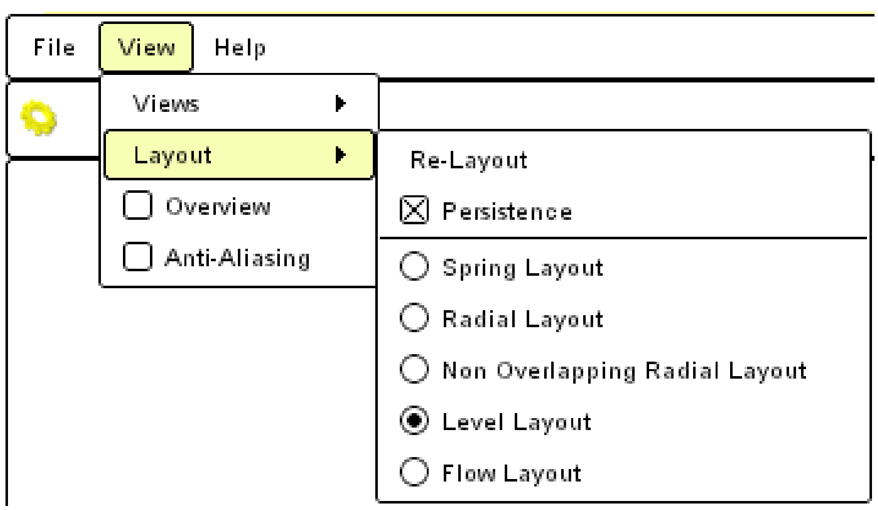


Figure 9.2: Layout Algorithms

The Tool Bar : contains icons representing various components that can be created in a graph canvas. The icons may not be present if the selected view does not support editing.

The View Display : is the main panel where each view renders graphs.

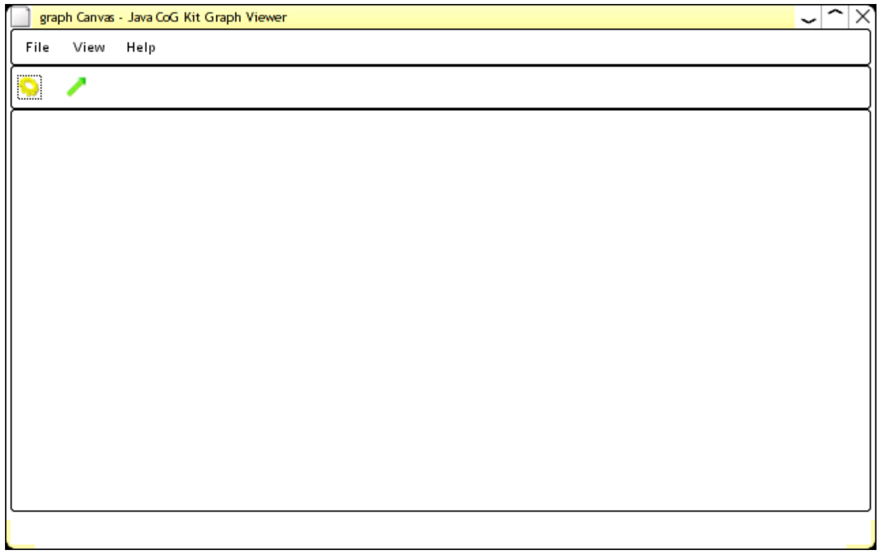


Figure 9.3: An Empty Graph Editor Window

When a graph is loaded or created, each component can have a set of options/actions accessible through a context menu. A snapshot of such a menu can be seen in Figure 9.4.

9.3.2 The HTML Target

The HTML target is used to render an HTML file that displays a graph. In order to be able to properly display a generated HTML graph, your browser will need to support JavaScript, and transparent PNG images. The HTML target has been tested with Mozilla 1.4 and up.

The output is optimized to produce a relatively small amount of data. For each distinct node icon, a separate image will be generated, but identical icons will not result in multiple images. There is also a certain amount of optimization involved in the generation of edges (arrows). An exponential scale is used to snap the dimensions of the generated arrow images, such that arrows having dimensions which differ by a small percentual ratio, will be represented by the same image. However this image will be further scaled by the JavaScript code in the html file as needed. This may result in aliasing and distortion of the rendered page, but it is a required compromise.

The available options for the *HTML* target are:

html.outputdir : The output directory where the html source and images will be generated.

html.graphview.layoutengine : Specifies the layout engine to be used when rendering a graph. The value is a fully qualifies Java class name. The predefined layout engines that the graph editor provides are located in the `org.globus.cog.gui.grapheditor.canvas.views.layouts` package. Available layout class names are:

`ExtendedSpringLayout` : A spring layout that does some initial heuristic layouting to reduce the overall layouting time. This layout will also skip the springing part for

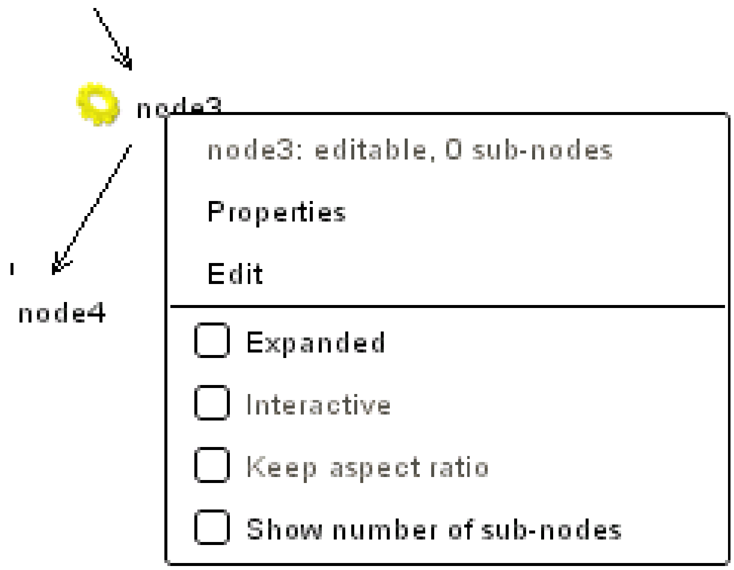


Figure 9.4: The Node Context Menu

large graphs (since the spring layout is $O(n^2)$).

`LevelLayout` : An $O(n)$ algorithm which places vertices on vertical levels using the edges as a factor of decision. Generally, if vertex *a* has an outgoing edge to vertex *b*, the later will be placed on a level below the former.

`RadialLayout` : Another $O(n)$ algorithm which tries to distribute edges for a vertex in such a way that the angles between consecutive edges will be the same.

`NonOverlappingRadialLayout` : This algorithm is a variation of the `LevelLayout`, with the distinction that instead of levels, the vertices are placed on concentric circles. While it tries not to overlap edges, it does not always succeed.

`Flow Layout` : $O(n)$ algorithm suitable for flow networks. It associates bounding boxes to vertices and, traversing the graph, resizes those bounding boxed according to the sizes of the bounding boxes of connected vertices.

9.3.3 The PostScript Target

Renders the graph in Encapsulated PostScript format.

Available options are:

postscript.outputdir : The directory where the output file will be placed.

postscript.outputfile : The name of the file that the output will be written to.

postscript.graphview.layoutengine : The layout engine used to render the graph. Accepts the same values as *html.graphview.layoutengine*9.3.2

9.3.4 The Remote Target

Forwards API calls to a remote graph editor service.

Available options are:

remote.contact : A *host:port* pair that represents the location of the graph editor service that the target will try to connect to.

9.4 Graph file format

The graphs are stored in a simple XML format. The simplest graph can be specified as follows:

```
<graph>
</graph>
```

Adding nodes can be done using the ¡node¿ element:

```
<graph>
    <node nodeid=”1” name=”the first node”/>
    <node nodeid=”2” name=”the second node”/>
</graph>
```

Edges can be added using the ’nodeid’s as references:

```
<graph>
        <node nodeid=”1” name=”the first node”/>
        <node nodeid=”2” name=”the second node”/>
        <edge from=”1” to=”2”/>
</graph>
```

Hierarchical graphs can be created too. In such graphs, each node can itself contain other graphs:

```
<graph>
        <node nodeid=”1” name=”the first node”>
                <node nodeid=”sn1” name=”subnode 1”/>
                <node nodeid=”sn2” name=”subnode 2”/>
                <node nodeid=”sn3” name=”subnode 3”/>
                <edge from=”sn1” to=”sn2”/>
                <edge from=”sn2” to=”sn3”/>
                <edge from=”sn3” to=”sn1”/>
        </node>
        <node nodeid=”2” name=”the second node”/>
        <edge from=”1” to=”2”/>
</graph>
```

Properties are specified as XML attributes. There are a few predefined properties that each node can have:

name : (String) appears as the text in the label used to render the node

iconfile : (String) an absolute path to an image that will appear as an icon for the rendered node overlayfile : (String) an absolute path to an image that will be overlayed on top of the base icon.

hue : (Float) specifies an additive adjustment for the hue of the icon. Changing this can shift the colors of the icon.

saturation : (Float) a multiplicative adjustment for the saturation of the colors in the icon.

value : (Float) a multiplicative adjustment for the value of the colors in the icon

status : (Integer) a value, ranging from 0 to 3 with the following meanings:

```
0 - stopped
1 - running
2 - failed
3 - completed
```

This will adjust the HSV color properties of the icon, and change the overlay, to give a visual representation of a possible state of a task.

You can also add your own custom properties (which will show up in the properties list for a node/edge):

```
<graph>
        <node
            nodeid="1"
            name="the first node"
            iconfile="/usr/share/icons/myicon.png"/>
        <node
            nodeid="2"
            name="the second node"
            myproperty="myvalue"/>

        <edge from="1" to="2"/>
</graph>
```

9.5 API

At the basis of the viewer/editor stands the `org.globus.cog.util.graph.Graph` class. For details, please consult the JavaDoc available at ??[1].

The editor can render nodes and edges that correctly implement the `org.globus.cog.gui.grap` interface, respectively, the `org.globus.cog.gui.grapheditor.edges.EdgeComponent` Generic implementations of the said interfaces are available at `org.globus.cog.gui.graphed` and `org.globus.cog.gui.grapheditor.generic.GenericEdge`.

Graphs are displayed in graph canvases, which in turn can have various views, used to render the graphs in particular ways. Views can also have transformations, used to algorithmically modify graphs, just before they are being displayed. These transformations can also be chained.

The nodes can also contain canvases, which in turn can contain other graphs, in a recursive manner. This allow for the effective use of hierarchical graph structures.

The following example shows how to build a simple graph and display it in a window:

[1] We need to have the JavaDoc of the CoG online

```
//create the root node
RootNode root = new RootNode();

//create a canvas for the root node
GraphCanvas canvas = root.createCanvas();

//create a graph structure
Graph graph = new Graph();

//create three node components
GenericNode genericNode1 = new GenericNode();
GenericNode genericNode2 = new GenericNode();
GenericNode genericNode3 = new GenericNode();

//add the components to the graph, and keep references
//to the node objects
Node node1 = graph.addNode(genericNode1);
Node node2 = graph.addNode(genericNode2);
Node node3 = graph.addNode(genericNode3);

//create two edge components
GenericEdge genericEdge1 = new GenericEdge();
GenericEdge genericEdge2 = new GenericEdge();

//add the edge components to the graph
//the first edge will go from node1 to node2,
//while the second one will go from node1
//to node3
graph.addEdge(node1, node2, genericEdge1);
graph.addEdge(node1, node3, genericEdge2);

//tell the canvas what graph it is supposed to deal
//with
canvas.setGraph(graph);

//choose a view for the canvas
canvas.setView(new GraphView());

//create a frame that displays everything
GraphFrame frame = new GraphFrame(root, false, 0);

//activate the frame
frame.activate();

//start the main loop
frame.run();
```

Node components and edge components (in short, graph components) can have properties. These properties can be used to change the appearance or behavior of the components. A list of meaningful properties for a GenericNode can be found in Table 9.4. The `nodeComponent.getPropertyValue(String name)` and `nodeComponent.setPropertyValue(String name, Object value)` methods can be used to query/modify the value of a property programmatically. The property changes will show immediate results on the screen. The example below will change the icon used for a node, and desaturate it:

```
genericNode1.setPropertyValue("iconfile", "/tmp/myicon.png");
genericNode1.setPropertyValue("saturation", new Float(0.1));
```

9.6 Scalability

When using the viewer with large graphs, please note that each element (node or edge) will take a total of about 1.3KB of memory when fully rendered. This means that a 50,000 node 150,000 edge graph will consume a total of about 300MB.

10 Portlet

Portals provide a secure, single point of interaction with diverse sources of information, personalized to the users needs.

Portlets are user-facing web application components, that can be managed, and personalized through a portal deployment.

The 'portlet' module provides a repository for portal enabled client-side web-clients: Portlets. A Grid Service Provider / Virtual Orginization (VO) can install these portlets, within a VO Portal/portlet container, so that users within the VO can Submit, View the status and output(s) of, and Administer their grid-tasks.

A brief description of portlets/remoteExecution: This portlet is the 'Grid Job Submission Portlet', and it allows users of the portal to submit Grid jobs. The Jobs submitted can be batch jobs, or redirected ones, with redirection of Standard Input, Standard Output and Standard Error. Once submitted, the 'User Task Set Management Portlet' can be used to manage the jobs.

A brief description of portlets/setManage: Common tasks supported by grid envionments are Job Submission, File Transfer, and Information Retreival. This portlet, the 'User Task Set Management Portlet', allows the management of user tasks, submitted to the User Task Set management service. The list of currently submitted tasks, and their output/error/status be viewed, and managed.

A brief description of portlets/taskSetUpload: This portlet allows for upload of a collection of, possibly unrelated, tasks as part of a Task Set, containing a list of task description(s) in an xml form. Once submitted, the 'User Task Set Management Portlet' can be used to manage the task(s).

10.0.1 Deployment

The build/deployment mechanism within the portlet module is as described in the README.txt in the cog home.

For ease of use and extensibility, the cog build model has been extended to the development, build, and deployment of individual portlets. Portlets are created and managed in the 'portlets' directory, within the portal module.

Running 'ant -projecthelp' in the concerned portlet/portlets/¡portlet-name¿ directory will show the supported ant targets, for that portlet.

A template portlet project is available in the 'portlets' directory.

10.0.2 Directory structure of portlet module

The directory structure of the portlet module is defined as follows:

CHANGES.txt	Changes to the module.
TODO.txt	Todo's for the module.
build.xml	And targets to build the module.
dependencies.xml	The dependencies of the module; currently on core.
etc	Jar manifest files.
launchers.xml	No Launchers at the moment.
lib	Jars to be shared amongst all the portlets.
portlet.xml	Master build file for the portlets.
portlets	Portlets repository directory.
portlets/<portlet-name>/build.xml	Ant build script.
portlets/<portlet-name>/conf	Jetspeed portlet registration files.
portlets/<portlet-name>/etc	Jar manifest files and Log4J configuration files.
portlets/<portlet-name>/lib	The jar files needed by the portlet.
portlets/<portlet-name>/project.properties	Portlet deployment properties. Have a look at the file in template for more information.
portlets/<portlet-name>/src	The portlet sources.
portlets/<portlet-name>/templates	Velocity templates for the portlet.
project.properties	Module properties.
src	Common portlet sources.

10.0.3 Prerequisite - Jetspeed

1. Download and install a Java Development Kit.

   To install and develop new portlets for Jetspeed, you have to first download and install the JDK as documented in Section ... if you have not done this so far. [1]

2. Download and install the Apache Ant build scripting environment.

   - This is required for both, building new portlets and, deploying the existing and new portlets.

   - Please use version 1.5 and above.

   - You can find documentation at: http://ant.apache.org/manual/index.html

3. Download and install the Servlet Container that will host the Jetspeed Portal web-application.

   - This relsase has been tested on Jakarta-Tomcat 4.1.24.

   - You can find documentation at: http://jakarta.apache.org/tomcat/tomcat-4.1-doc/index.html

   - A quick start procedure to get Jakarta-Tomcat 4.1.x installed is:

   1) Download the jakarta-tomcat binary installation archive from: http://apache.get-software.com/jakarta/tomcat-4/v4.1.29/bin/

[1] GvL: please complete the apropiate section, this used to be here before: This release has been tested on Sun JDK 1.4.1 and above.; You can find documentation at: http://java.sun.com/

2) Unzip the contents of the tarball into a chosen location.

3) Setup the port for the http connector in the server.xml file, usually found at: <tomcat-install-dir>/conf/server.xml

4. Download and install the Apache Jetspeed portal implementation.

   - This release has been tested on Jetspeed 1.4b3.

   - You can find documentation at: http://jakarta.apache.org/jetspeed/site/install.html

   - A quick start procedure is:

   1) Download the jetspeed WAR file from: http://jakarta.apache.org/builds/jakarta-jetspeed/release/v1.4b3/

   2) Unzip its contents (the jetspeed.war file) into the 'webapps' directory of your chosen Servlet Container (compatible with either the Servlet 2.2, or 2.3 API specification).

   3) Point your browser to your installation, to deploy the jetspeed webapplication. This is usually: http://<ip>:<port>/jetspeed

10.0.4 Prerequisite - MyProxy

1. Download and install the My Proxy Manager portlet. This portlet is required for retreiving proxy grid credentials, previously stored in a MyProxy credential repository.

   - Currently we support the 2.0 version of the 'Proxy Manager' portlet.

   - You can get this portlet source from the Alliance Portal site, hosted at Extreme! Computing Labs, at: http://www.extreme.indiana.edu/xportlets/project/release/ , in the package 'xportlets-proxymanager-2.0-src.tar.gz'. - A quick start procedure is:

   1) Download the 'xportlets-proxymanager' release archive file, and untar the distribution into a temporary directory.

   2) Copy the 'build.properties.template' file, in the xportlets- -proxymanager, to 'build.properties', and edit the 'alliance.home' ant property to point to the jetspeed installation.

   - For a jakarta-tomcat servlet container, this would be: <tomcat-install-dir>/webapps/jetspeed/

   3) Run the 'ant build' and then the 'ant deploy' command in the xportlets-proxymanager directory. This will build and deploy the portlet into the jetspeed installation.

10.0.5 Configuration

1. Please configure the Java CoG Kit, installation with the required certificates and property files. More information in this regards can be found in the Java CoG Kit manual, found here: http://www.globus.org/cog/manual-user.pdf

   The following components need to be installed, on the portal deployment machine: - Ensure that you have installed the appropriate Certification Authority (CA) certificates.

- Ensure that you have your user-certificates installed.

- Ensure that the /.globus/cog.properties is properly configured.

- You need to have a valid proxy-certificate.

2. Please ensure that you set the following ant properties, in the 'project.properties' file in the respective portlet directory.

   a) 'jetspeed.home': the complete path to the jetspeed web application installation. - For a jakarta-tomcat servlet container, this would be: <tomcat-install-dir>/webapps/jetspeed/

   b) 'servlet.lib.dir': the complete path to where 'servlet.jar' can be found. - For jakarta-tomcat version(s) 3.3.x this jarfile can be found in, and thus the property would be set to: <tomcat-install-dir>/lib/common/

   - For jakarta-tomcat version(s) 4.x this jarfile can be found in, and thus the property would be set to: <tomcat-install-dir>/common/lib/

   - For other web-application containers please check their distributions for this jar file.

10.0.6 Installation

1. Running the command 'ant dist', in the directory: portlet/portlets/<portlet-name>/ , will compile, build, and deploy the portlet into the jetspeed installation.

   - The available portlets are:

   1) the 'Grid Job Submission Portlet'

   2) the 'User Task Set Management Portlet', and

   3) the 'TaskSet Upload Portlet'

   - For further details on the build system of the portlets/ template, do have a look at the README.txt in the cog home directory.

2. Use the jetspeed portlet listing/management interface, per user to add the portlets to the users workspaces.

11 Java CoG Kit QoS Module

The QGS is the QoS an enhanced provider that is also available within the Grid QoS Management framework (G-QoSM), and, currently, supports access to Grid resources with QoS guarantees. Two resouce allocation strategies are supported; a) time-domain, and b) resource-domain. Time-domain requests that is exclusive access to computation resource of a Grid node, which entails the user to have full access to the computer resource where the QGS service is installed, then the user can submit job(s) to this particular resource through out the defined period in the QoS agreement and no other users' jobs share this computation resource. Resource-domain request is request for computation resource in a shared mode, therefore, the applicatioin/client must provide a specific computation capacity requirment, e.g a percentage of the overallCPU capacity. Then the application/client can submit jobs, during the specifed duration in the agreement, to the Grid node, and jobs will be executed under the that specific CPU QoS constraints.

11.0.7 QGS Installation Prerequists

Make sure the following components are properlty installed and configured:

1. Globus toolkit 3.0 or later versions – full installation or the 'core'
2. Java CoG kit 1.1a or later versions is propery installed and configured
3. Dynamic Soft Realtime scheduler (DSRT), available with this distribution – make sure you use the DSRT with this distribution as it has some customized API
4. Edit the file "config.txt" available in the root directory of this distribution, with the DSRT installation path and save the file in the '.globus/' directory.
5. Java VM and apache ant installed.

11.0.8 QGS Compilation and service deployment

If you decided not to compile the QGS and generate jar and gar files, go to setep (3) ,,,, and use the supplied gar and jar files available in the /qos/lib directory.

1. Edit the file 'build.properties' in the installation root directory with the right value of the 'ogsa.root', which should be set to the ogsa installation.
2. From the installation directory, run the convinient script, created by the GT3 team, as shown below, to compile the QGS service and create the appropoeriate jar and gar files.

   ./compileService.sh org/globus/cog/qos/server/imple/Qos.java

   If all goes well, then you should have a build directory with all the jar, gar and the compiled classes.

3. Create a proxy, if you don't have a valid one; one way to do this is from the CoG_dir/bin enter the following:

   ./visual-proxy-init

4. From the ogsa installation directory deploy the service by entering the following command:

   ant deploy -Dgar.name=$QGS_DIR/src/build/lib/org.globus.cog.qos.server.Qos.gar

   where QGS_DIR is the installation directory of this distribution

5. Sart the ogsi container by entering the follwoing command from the ogsa directory:

   ant startContainer

6. Create a presistant instance of the QGS by entering the following command from the ¡ogsa_dir¿/bin:

   ogsi-create-service http://localhost:8080/ogsa/services/org/globus/cog/qos/server/QosService qos

   this should be entered as one command.

7. To ensure that the service instance has been started, from the ogsa_dir enter the following command:

   ant gui

   This command starts the ogsi visual brower.

   You should see in the broweser: 'A QoS Service Factory' and 'A QoS Service Instance' with both are in 'ACTIVE' states

8. If all goes well, and you can see the service instance in the browser as 'ACTIVE', then Congrantulation!! the QGS is deployed and instantiated correctly.

9. The QGS needs to know which gatekeeper to submit jobs to, therefore, we suggest that you start a jglobus gatekeeper availble within the CoG installation directory and get its contact informatin. This can be done as follows:

   ./globus-personal-gatekeeper

   this command will return a contact information. You you have a gram service running, you can naturally you use it instead onf jglobus gatekeeper.

   when the gram service contact inofrmation is obtained, run a launcher program available in the /dist/bin as follows:

   ./qos-adminClient ¡QGS url¿ ¡Gram contact¿

10. Now you should be able to use the QoS service provides by the QGS. You may start with exeuting some or the supplied examples or interact with the QGS using the supplied GUI interface

11.0.9 Examples

There are a number of examples in the package org/globus/cog/qos/examples where you can take a look at how the client can be accessed. The launchers of these examples are available in the /dist/bin directory.

You can also interact with the QGS service using our GUI client, which can be run from the from the /dist/bin directory – the launcher script name is ./qos-guiClient

```
C:\cygwin\home\laszewsk\work\cog\modules\qos\dsrt\index.html
```

11.1 Resources

11.2 Common

11.3 All

12 Command Tools

13 grid-cert-request

The certificate management module is a set of tools that make it easier for users to manage their certificates. For instance there are tools to generate a certificate request, store credential on MyProxy server, view local credential, renew certificates and revoke certificates.

Administrators have to choice of deploying these tools as signed Java Applets and/or as signed Java WebStart Applets.

The benefits of signed Java Applet are integration into web pages however they required Java capable browsers. Java singed WebStart Applets do not have this requirement. The WebStart mechanism also has the advantage of caching the jars used by the applets. One disadvantage to the WebStart mechanism is that the Applet will not integrate into web page.

Both of these deployment methods are appealing because they dont require any installation of OGSA or CoG by the client. The deployment mechanism will install the necessary jars to run the certificate management tools.

This module consists of .java files for the actual applets and of .jsp and .html pages to launch the Java Applets and .jnlp files to launch the WebStart Applets. These files contain parameters that can be changed by an administrator to change such things as the Certificate Authority, MyProxy server location and background color of the applet.

These applets need to be signed

(explain how this is done)

A user must trust the entity that signed the applets.

[warning.jpg]

At the moment the jce jar is not signed properly by Bouncy Castle. Once we use the latest jar it will be ok but for the moment you will get this dialog. Simply press the X not the abort button.

[jce.jar]

Once the user grants the applets security access it will start.

The first step a new grid user will have to do is get credentials. To do this you need to generate a certificate request and have it signed by your certificate authority. Use the CertReqApplet to do this:

[certreq.jpg]

(description of params goes here)

Once you get a response from your CA place it in your usercert.pem as described in the email. You can now use the certificate info applet to see your local credentials.

[cerinfo.jpg]

Supposing you were going away on a business trip you may want to put some temporary credential on a MyProxy server. See myproxy command line tool for more details on what this means.

[myproxy.jpg]

(description of params goes here)

After a certain amount of time your credential will expire. Before this happens your CA will send you a renewal notification with a challenge phrase. You can use the certificate renew applet to generate your renewal request.

```
[certrenew_applet.jpg]



(description of params goes here)
```

If for some reason your credentials get compromised or you simply dont need them anymore you may want to destroy your credentials and notify your CA that you did so. The certificate revocation applet can be used to delete the certificate files and notify your CA.

```
[certrevocation_appet.jpg]



(description of params goes here)
```

grid-cert-request [-help] [options ...]

SYNOPSIS.

grid-cert-request can create user, host, and LDAP server certificates. A certificate request and private key will be created.You will be asked to enter a PEM pass phrase. This pass phrase is akin to your account password, and is used to protect your key file. If you forget your pass phrase, you will need to obtain a new certificate.

EXAMPLES.

```
 grid-cert-request
         - Creating a user certificate

    grid-cert-request -host [my.host.fqdn]
          -  Creating a host or gatekeeper certifcate

    grid-cert-request -service ldap -host [my.host.fqdn]
          -   Creating a LDAP server certificate
```

OPTION

```
    -version           : Display version
    - , -h, -help,     : Display usage
```

```
-usage
-cn <name>,         : Common name of the user
-commonname <name>
-service <service> : Create certificate for a service. Requires
                     the -host option and implies that the gene
                     key will not be password protected (ie imp
-host <FQDN>        : Create certificate for a host named <FQDN>
-dir <dir_name>     : Changes the directory the private key and
                     request will be placed in. By default user
                     certificates are placed in /home/user/.glo
                     certificates are placed in /etc/grid-secur
                     service certificates are place in
                     /etc/grid-security/<service>.
-prefix <prefix>    : Causes the generated files to be named
                     <prefix>cert.pem, <prefix>key.pem and
                     <prefix>cert_request.pem
-nopw,              : Create certificate without a passwd
-nodes,
-nopassphrase,
-verbose            : Don’t clear the screen <<Not used>>
-int[eractive]      : Prompt user for each component of the DN
                     <<Not implemented yet>>
-force              : Overwrites preexisting certifictes;
```

Bibliography

[1] G. von Laszewski and P. Wagstrom, *Tools and Environments for Parallel and Distributed Computing*, ser. Series on Parallel and Distributed Computing. Wiley, 2004, ch. Gestalt of the Grid, pp. 149–187. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--gestalt.pdf 8

[2] G. von Laszewski and K. Amin, *Grid Middleware*. Wiley, 2004, ch. Middleware for Commnications, to be published. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-gridmiddleware.pdf 8

Index

Acknowledgments, 13
Administrative Contact, 13

Bugs, 9

Contact, 13

Karajan, 55
  Methods
    allocateHost, 67, 77, 79
    checkpoint, 75, 77
    default, 65, 77
    echo, 62, 68, 77
    elementDef, 77
    executeJava, 68, 78
    for, 64, 78
    foreach, 64, 78
    function:contains, 69, 78
    function:numberFormat, 69, 78
    function:readFile, 69, 79
    function:UID, 69, 79
    generateError, 68, 79
    gridExecute, 66, 74, 77, 79
    gridTransfer, 66, 75, 77, 79
    ignoreErrors, 68, 80
    include, 68, 71, 72, 75, 80
    javaBean, 69, 80
    javaBean>invokeVoid, 69, 80
    javaBean>setProperty, 69, 80
    nonCheckpointable, 80
    parallel, 62, 64, 81
    project, 68, 81
    restartOnError, 68, 81
    scheduler, 67, 75, 81
    scheduler>grid, 65, 75, 81
    scheduler>grid>host, 65, 81
    scheduler>grid>host>service, 65, 68, 81, 82
    scheduler>taskHandler, 68, 82
    scheduler>taskHandler>securityContext, 68, 82
    scheduler>taskHandler>securityContext>property, 82
    sequential, 62, 82
    setvar, 60, 82
    template, 65, 82
    templateDef, 65, 82, 83
    wait, 69, 83

License, 14
  bouncycastle, 24
  cryptix, 24
  Globus Toolkit, 16
  GPTL, 16
  Java CoG Kit, 20
  JCoGPL, 20
  junit, 24
  log4j, 24
  puretls, 24
  soaprmi11, 24
  xerces, 24
  xml4j, 24

Mailing List, 10

Project Registration, 14

Website, 9
Workflow, 55
  Karajan, 55