

Java CoG Kit Workflow Guide

From Java CoG Kit

Gregor von Laszewski and Mike Hategan

- Current version: 4.1.4, 08/17/2006
- URL: Java CoG Kit Workflow Guide
- Wiki Revision: [1] (http://wiki.cogkit.org/index.php?title=Java_CoG_Kit_Workflow_Guide&oldid=)

Previous version: 4.1.3

(http://wiki.cogkit.org/index.php?title=Java_CoG_Kit_Workflow_Guide&oldid=4098)

For more information contact gregor@mcs.anl.gov and hategan@mcs.anl.gov

For support, please use our bugzilla system

(<http://www.cogkit.org/wiki/cog/moin.cgi/CoGKit/Bugs?action=print>)

Contents

- 1 Abstract
- 2 Summary of Changes to Previous Versions
- 3 Install for Grid Users
- 4 Install for non Grid Users for the Impatient
- 5 Hello world!
- 6 Variables
- 7 Tasks/Jobs
 - 7.1 A local job
 - 7.2 Authentication to a remote resource
 - 7.2.1 grid-proxy-init
 - 7.2.2 SSH Authentication
 - 7.3 A remote job
 - 7.4 Retrieving the output
 - 7.4.1 Getting stdout
 - 7.4.2 Getting stderr
- 8 Services and Providers
 - 8.1 GT4 - Classic/GT2
 - 8.2 GT4 - WSRF-GRAM
 - 8.3 GT3.2.1
 - 8.4 SSH
 - 8.4.1 Using a noninteractive security context with SSH providers
- 9 Concurrency
 - 9.1 The parallel tag
 - 9.2 The sequential tag
- 10 Modularization
 - 10.1 Defining elements
 - 10.2 Proposed enhancements: Namespaces
 - 10.3 Proposed enhancements: Scoped Includes
- 11 Lists
- 12 Return values
- 13 Control Structures
 - 13.1 Loops
 - 13.1.1 while

- 13.1.2 for
- 13.1.3 Example
- 13.2 Condition
- 14 Forms (Swing integration)
- 15 System Command
- 16 Invoking Java methods with the Java Library
- 17 Other Available Features
- 18 Limitations
- 19 Appendix
- 20 References
- 21 Current Tasks

1 Abstract

This page contains a draft for the Java CoG Kit workflow guide. Once the information on this page is collected and complete we will include it in the Guide. We are working currently on the Java CoG kit version 4.1.4. This version has some significant changes towards the older version that is documented in the workflow guide PDF (<http://www.cogkit.org/release/4.1.4/manual/workflow.pdf>) HTML (<http://www.cogkit.org/release/4.1.4/manual/workflow/workflow.html>) .

Sections marked with **Proposed enhancements** document features that are under discussion. They are not part of the current software nor is sure that they will be. Please use the discussion section to voice your oppinions. Please also let us know if you need other features and enhancements. These proposed enhancements are discussed in our discussion section (http://www.cogkit.org/w/index.php/Talk:Java_CoG_Kit_Workflow_Examples) .

Warning: This documentation is reflecting the version 4.1.4 of the CoG Kit and newer. It is not compatible with the Version 4.0.1. The version 4.1.4 is only available from the CVS.

2 Summary of Changes to Previous Versions

- `<if><condition> ... </condition> </if>` do no longer contain the `<condition> ... </condition>`.
Rationale: It is clear that an if is followed by a Boolean expression. This also allowed us to develop a simplified functional programming language construct `if(boolean, doWhenItsTrue, doWhenItsFalse)`.
- We have renamed the file `cog.xml` to `cogkit.xml`.

3 Install for Grid Users

First you have to download the Java CoG Kit version 4 and install it on your client. The vaious ways to do this are documented in the Instalation Guide. If you follow the link to the download (<http://www.cogkit.org/php/download.html>) area, you will find the installation instructions. For the purpose of thsi documentation we use the CVS installation as described in Chapter 9 (Development Distribution).

Once you have the code from CVS, you need to compile it.

```
cd src/cog
ant dist
```

This will create in the dist dir the Java CoG Kit distribution.

Please, add the <yourpath>dist/cog-4.1.4/bin to your path environment variable.

Now you have to let Java CoG Kit know where your certificates are. The easiest way to configure this is to use the ./cog-setup tool

```
cog-setup
```

Once you have done this you can call the Java CoG Kit Workflow from the command line. In case you have no Grid available you still can use the workflow, but some of the commands and examples in this guide will not work. You need to simply skip the cog-setup step.

Please test it with the following commands after you downloaded the helloworld.xml (<http://www.cogkit.org/viewcvs/viewcvs.cgi/src/cog/modules/karajan/examples/helloworld.xml?rev=HEAD&example> from our CVS examples (<http://www.cogkit.org/viewcvs/viewcvs.cgi/src/cog/modules/karajan/examples/>)

```
cd ../dist/cog-4.1.4/examples/karajan
cog-workflow helloworld.xml
```

If this does not work something is wrong with your setup.

4 Install for non Grid Users for the Impatient

Although we believe our documentation on the previous section is complete we have provided on popular demand this section for the impatient. We assume you have Java, ant, and wget properly installed on your system.

Now follow these steps, where > indicates a shell prompt.

1. If you have not done so far go to <http://www.cogkit.org/register>
2. > cvs -d:pserver:anonymous@cvs.cogkit.org:/cvs/cogkit checkout src/cog
3. > cd cogkit
4. > cd src/cog
5. > ant dist
6. > cd dist/cog-4_1_4/bin
7. > wget http://svn.sourceforge.net/viewcvs.cgi/*checkout*/cogkit/trunk/examples/karajan/echo.xml (If you do not have wget, you can use your favourite browser and store the file in the bin dir.)
8. > ./cog-workflow echo.xml

the output will be

```
Hello world!
```

Please note that xml like workflows must be stored in a file with the ending .xml

5 Hello world!

It is relatively straightforward what the "Hello World" program does. What is of relevance is that Karajan workflows are written inside a **<project>** element. The other relevant part is the **<include file="cogkit.xml"/>** line. It instructs Karajan to load the specified file inside the project and interpret it. The file **cogkit.xml** contains the definitions for advanced language constructs including parallelism.

```
<project>
  <include file="cogkit.xml"/>
  <echo message="Hello world!"/>
</project>
```

Download: [echo.xml](#)

(<http://svn.sourceforge.net/viewcvs.cgi/cogkit/trunk/examples/karajan/echo.xml?view=markup>)

6 Variables

Variables, similar to other languages, are symbolic names that represent values. The binding of a value to a name in Karajan is done using the **<set>** element:

```
<project>
  <include file="cogkit.xml"/>
  <set name="hello" value="Hello World!"/>
  <echo message="{hello}"/>
</project>
```

Download: [set.xml](#)

(<http://svn.sourceforge.net/viewcvs.cgi/cogkit/trunk/examples/karajan/set.xml?view=markup>)

The values of variables can be retrieved using curly brace expansion. Whenever a name is put inside curly braces, Karajan will look for a variable with that name, and substitute its value.

7 Tasks/Jobs

The Java CoG Kit introduces the model of tasks. In a Karajan workflow tasks are implemented as processes that are managed by the Karajan Workflow engine, but are executed externally. They can be either executed locally, on the machine Karajan is running, or remotely, on other machines.

7.1 A local job

The following example will help you to execute a job on your local machine (we assume you use a *nix machine):

```
<project>
  <include file="cogkit.xml"/>
  <execute executable="/bin/sleep" arguments="10" provider="local"/>
  <echo message="Job completed"/>
</project>
```

7.2 Authentication to a remote resource

7.2.1 grid-proxy-init

Use grid-proxy-init for GT2, GT3, and GT4. We assume you have a valid certificate and placed it according to the Globus Toolkit in the .globus directory. The details regarding initializing your proxy can be viewed here:

- Command_Line Tool
 - GUI Tool
- Show how to do grid-proxy-init form the commandline (Deepti)
 - Show how to do vizual-grid-proxy-init with screenshot use albert as username (Deepti)
 - Show how to do an ssh authentication for the use with ssh (Deepti)

7.2.2 SSH Authentication

Using Karajan, the authentication can be done as described here. Otherwise

7.3 A remote job

Karajan makes it relatively easy to use other machines, too:

```
<project>
  <include file="cogkit.xml"/>
  <execute executable="/bin/sleep"
          arguments="10"
          host="hot.mcs.anl.gov"
          provider="GT2"/>
  <echo message="Job completed"/>
</project>
```

The difference from the local example consists of the addition of a **host** argument to **<execute>** which indicates the remote machine the executable, and a **provider** attribute, which tells Karajan how the job is to be submitted. Which providers are supported are discussed in the Java CoG Kit Abstraction Guide (http://www.cogkit.org/user/index.php/Java_CoG_Kit_Abstraction_Guide) . At this time we recommend to use GT2, GT4, and SSH.

7.4 Retrieving the output

In the next example we will gradually add features to outline some of the abilities of the Karajan workflow. We start with the example form the previous section and add a message to the screen that indicates its completion.

```

<project>
  <include file="cogkit.xml"/>
  <execute executable="/bin/date"
            host="hot.mcs.anl.gov" provider="GT2"/>
  <echo message="Job completed"/>
</project>

```

Although running the above example you will probably notice the *Job Completed* message on the screen, it will not list the date and time. This happens because the output from a job is not automatically redirected to the local machine. Controlling what happens to the output is left to the user as it adds flexibility to our framework.

7.4.1 Getting stdout

Retrieving the output of a command consists of two steps. First **<execute>** must be told to redirect the output of the command to a file, then the file needs to be transferred back to the local host:

```

<project>
  <include file="cogkit.xml"/>
  <execute executable="/bin/date"
            stdout="thedata"
            host="hot.mcs.anl.gov" provider="GT2"/>
  <echo message="Job completed. Transferring the output"/>
  <transfer srchost="hot.mcs.anl.gov" srcfile="thedata"
            desthost="localhost" provider="gridftp"/>
  <echo message="Transfer complete"/>
</project>

```

With this program the date program can be invoked on the remote computer and the output can be transferred back. However, it will still not be displayed on the screen. Before that is done, a new concept must be introduced.

In order to display the contents of a text file, it must be read first, and the contents put inside a variable. We declare for our purpose the variable with the name `date` in which we dump the contents of the file called "thedata". Thus, we enhance our program as follows:

```

<project>
  <include file="cogkit.xml"/>
  <execute executable="/bin/date"
            stdout="thedata"
            host="hot.mcs.anl.gov" provider="GT2"/>
  <echo message="Job completed. Transferring the output"/>
  <transfer srchost="hot.mcs.anl.gov" srcfile="thedata"
            desthost="localhost" provider="gridftp"/>
  <echo message="Transfer complete"/>
  <set name="date">
    <readFile file="thedata"/>
  </set>
  <echo message="The date is {date}"/>
</project>

```

In case you like to use a different provider, please consult our abstraction guide. We recommend that you use GT2, GT4, or SCP. ⚠ Mike is this right?

7.4.2 Getting stderr

Similar to the standard output of the executable, the standard error can be retrieved.

```

<project>
  <include file="cogkit.xml"/>
  <execute executable="/bin/ls" arguments="-al"
            stdout="stdout" stderr="stderr"
            host="hot.mcs.anl.gov" provider="GT2"/>
  <echo message="Job completed. Transferring stdout and stderr"/>
  <transfer srchost="hot.mcs.anl.gov" srcfile="stdout"
            desthost="localhost" provider="gridftp"/>
  <echo message="Stdout transferred"/>
  <transfer srchost="hot.mcs.anl.gov" srcfile="stderr"
            desthost="localhost" provider="gridftp"/>
  <echo message="Stderr transferred"/>
</project>

```

We have omitted in this last example the code for displaying the contents of the files which is left as a trivial exercise for the reader.

This example leads to the question: why would the transfer of *stderr* have to wait for the transfer of *stdout* to complete? The answer is "it doesn't" as our next example shows.

8 Services and Providers

In the previous example, we have seen how simple it is to submit an executable and a file transfer as part of our task abstractions within the workflow. Now let us focus on generalizing this example a bit more to make it more adaptable. To do so we include variables for the host name and the provider type. We introduce here, the notion of different services/providers available under a single host. This is defined as below:

```

<project>
  <include file="cogkit.xml"/>
  <set name="provider" value="gt2"/>
  <set name="hot" value="hot.mcs.anl.gov"/>
  <host name="{hot}" cpus="2">
    <service provider="gt4.0.0" type="execution" uri="{hot}:4002"/>
    <service provider="gt2" type="execution" uri="{hot}:3952"/>
    <service provider="gt4.0.0" type="file-transfer" uri="{hot}:4001"/>
    <service provider="gt2" type="file-transfer" uri="{hot}:3951"/>
    <service provider="ssh" type="execution" uri="{hot}" securityContext="{ssh-doe}"/>
  </host>
  <execute executable="/bin/ls" arguments="-al"
            stdout="stdout" stderr="stderr"
            host="{host}" provider="{provider}"/>
  <echo message="Job completed. Transferring stdout and stderr"/>
  <transfer srchost="hot.mcs.anl.gov" srcfile="stdout"
            desthost="localhost" provider="gridftp"/>
  <echo message="Stdout transferred"/>
  <transfer srchost="{host}" srcfile="stderr"
            desthost="localhost" provider="gridftp"/>
  <echo message="Stderr transferred"/>
</project>

```

8.1 GT4 - Classic/GT2

In this example we demonstrate to submit to a Globus Toolkit GT2 GRAM service running on the default port on the machine hot.mcs.anl.gov with the assumption that the user starting this program is in the Grid map file. For your own experiments you should use a GT2 machine you have access to.

For convenience, we have stored the type of the provider in a variable. The reason for this is simple. Assume

a system administrator were to decide to upgrade the service to GT3, or GT4, than the user can with the change of the variable *type* access these services instead of using GT2. This is demonstrated in the next two examples.

```
<project>
<include file="cogkit.xml"/>

<set name="provider" value="GT2"/>
<set name="host" value="hot.mcs.anl.gov"/>

<execute executable="/bin/ls" arguments="-al"
          stdout="stdout" stderr="stderr"
          host="{host}" provider="{provider}"/>
<echo message="Job completed. Transferring stdout and stderr"/>
<transfer srchost="hot.mcs.anl.gov" srcfile="stdout"
          desthost="localhost" provider="gridftp"/>
<echo message="Stdout transferred"/>
<transfer srchost="hot.mcs.anl.gov" srcfile="stderr"
          desthost="localhost" provider="gridftp"/>
<echo message="Stderr transferred"/>
</project>
```

8.2 GT4 - WSRF-GRAM

In contrast to our GT2 example the only thing you have to change is the type of the provider to GT4. We assume that a GT4 job execution service is running on the machine.

```
<project>
<include file="cogkit.xml"/>

<set name="provider" value="GT4"/>
<set name="host" value="hot.mcs.anl.gov"/>

<execute executable="/bin/ls" arguments="-al"
          stdout="stdout" stderr="stderr"
          host="{host}" provider="{provider}"/>
<echo message="Job completed. Transferring stdout and stderr"/>
<transfer srchost="hot.mcs.anl.gov" srcfile="stdout"
          desthost="localhost" provider="gridftp"/>
<echo message="Stdout transferred"/>
<transfer srchost="hot.mcs.anl.gov" srcfile="stderr"
          desthost="localhost" provider="gridftp"/>
<echo message="Stderr transferred"/>
</project>
```

8.3 GT3.2.1

Note: GT3 is no longer supported.

~~Please note that we do not support officially GT3 any longer and it is provided as is. In case you need support for it you can subcontract us. Please also note that there are multiple providers for GT3 and therefore the full version is required (i.e. GT3.2.1). In contrast to our GT2 example the only thing you have to change is the type of the provider to GT3.2.1. We assume that a GT3.2.1 job execution service is running on the machine.~~

```

<project>
<include file="cogkit.xml"/>

<set name="provider" value="GT3.2.1"/>
<set name="host" value="hot.mcs.anl.gov"/>

<execute executable="/bin/ls" arguments="a1"
          stdout="stdout" stderr="stderr"
          host="{host}" provider="{provider}"/>
<echo message="Job completed. Transferring stdout and stderr"/>
<transfer srchost="hot.mcs.anl.gov" srcfile="stdout"
          desthost="localhost" provider="gridftp"/>
<echo message="Stdout transferred"/>
<transfer srchost="hot.mcs.anl.gov" srcfile="stderr"
          desthost="localhost" provider="gridftp"/>
<echo message="Stderr transferred"/>
</project>

```

8.4 SSH

The following example demonstrates an easy way on executing a remote job with the karajan workflow language while using the ssh provider instead of the Globus Toolkit provider. First, we include the necessary system and task elements. Next, we define a security context. In this case we ask interactively for the SSH passphrase. To specify in which way we contact the remote resource, we define it through the host command which we store in the variable myhost. Next we can reuse the resource specification to submit the task to it within the task:execute element.

```

<project>
  <include file="cogkit.xml"/>

  <set name="sc">
    <InteractiveSSHSecurityContext/>
  </set>

  <set name="myhost">
    <host name="somename">
      <service type="execution"
              provider="ssh"
              securityContext="{sc}"
              uri="host.com"/>
    </host>
  </set>

  <execute executable="/bin/date"
          host="{myhost}"
          redirect="true"
          provider="ssh"/>
</project>

```

8.4.1 Using a noninteractive security context with SSH providers

To use a non interactive security context one can use the following security context. To specify a private key or a passphrase use

```

<SSHSecurityContext>
  <publicKeyAuthentication username= privatekey= passphrase=/>
</SSHSecurityContext>

```

In case the private key is at a default location you can also use

```
<SSHSecurityContext>
  <passwordAuthentication username="yourname" password="yourpassword" />
</SSHSecurityContext>
```

This feature is documented in the workflow guide

(http://www.cogkit.org/viewcvs/viewcvs.cgi/*checkout*/doc/manual/guide/workflow/workflow.pdf?rev=HE).

9 Concurrency

9.1 The parallel tag

One way of expressing concurrency in Karajan is to use it in an explicit declarative fashion through the sequential and parallel tags. To utilize this capability, we like to transfer now the two files stdout and stderr in parallel.

```
<project>
  <include file="cogkit.xml" />
  <execute executable="/bin/ls" arguments="-al"
           stdout="stdout" stderr="stderr"
           host="hot.mcs.anl.gov" provider="GT2" />
  <echo message="Job completed. Transferring stdout and stderr" />

  <parallel>
    <transfer srchost="hot.mcs.anl.gov" srcfile="stdout"
             desthost="localhost" provider="gridftp" />
    <transfer srchost="hot.mcs.anl.gov" srcfile="stderr"
             desthost="localhost" provider="gridftp" />
  </parallel>
  <echo message="Stdout and stderr transferred" />
</project>
```

9.2 The sequential tag

Let us now try to find the time on two machines. Since the task of finding out the time on one machine is independent of the task of finding out the time on another machine, these two can run in parallel. However, it is still needed that the executable completes before its output can be transferred. This is achieved by introducing a new element, named **<sequential>**, which forces sequential execution. Naturally our two sequential blocks will be included in a parallel block indicating that the blocks are executed in parallel.

```

<project>
  <include file="cogkit.xml"/>
  <parallel>
    <sequential>
      <execute executable="/bin/date"
                stdout="date-hot"
                host="hot.mcs.anl.gov" provider="GT2"/>
      <echo message="Job 1 completed. Transferring output"/>
      <transfer srchost="hot.mcs.anl.gov" srcfile="date-hot"
                desthost="localhost" provider="gridftp"/>
      <echo message="Output 1 transferred"/>
    </sequential>
    <sequential>
      <execute executable="/bin/date"
                stdout="date-sunny"
                host="sunny.mcs.anl.gov" provider="GT2"/>
      <echo message="Job 2 completed. Transferring output"/>
      <transfer srchost="sunny.mcs.anl.gov"
                srcfile="date-sunny" desthost="localhost"
                provider="gridftp"/>
      <echo message="Output 2 transferred"/>
    </sequential>
  </parallel>
</project>

```

Unfortunately the above examples require a little too much duplicate code. How to simplify the code is shown in the section describing the modularization.

10 Modularization

10.1 Defining elements

In Karajan new elements can be defined for certain repetitive tasks. The concept is similar to the definition of procedures.

```

<project>
  <include file="cogkit.xml"/>
  <element name="date" arguments="host">
    <execute executable="/bin/date"
              stdout="date-{"host}" host="{host}" provider="GT2"/>
    <echo message="Job on {host} completed. Transferring output"/>
    <transfer srchost="{host}" srcfile="date-{"host}"
              desthost="localhost" provider="gridftp"/>
    <echo message="Output from {host} transferred"/>
    <set name="date">
      <readFile file="date-{"host}"/>
    </set>
    <echo message="The date on {host} is {date}"/>
  </element>
  <parallel>
    <date host="hot.mcs.anl.gov"/>
    <date host="sunny.mcs.anl.gov"/>
  </parallel>
</project>

```

The definition is done using the **<element>** Karajan element. The **arguments** attribute tells the definition what arguments to expect when invoked later. Since the elements inside the *date* definition are not inside a **<parallel>** container, the default sequential behavior will be assumed, without the need to explicitly include the elements inside a **<sequential>**.

A more comprehensive example with integration of Java Code is available [here (<http://www.cogkit.org/dev/index.php/Karajan/JavaElement>)].

10.2 Proposed enhancements: Namespaces

The Java CoG Kit contains a simple mechanism of conducting namespaces. It is defined as part of the include mechanism. The following example defines two files. The one file is called english.xml and the other file is called german.xml. Within these files we have defined a method hello that dependent on the namespace prints either out "Hello World" or "Hallo Welt". In addition we included the definition of a unique element in english.xml.

german.xml:

```
<namespace prefix="german">
  <element name="hello">
    <print message="Hallo Welt"/>
  </element>
</namespace>
```

english.xml:

```
<namespace prefix="english">
  <element name="hello">
    <print message="Hello World"/>
  </element>
  <element name="unique">
    <print message="Hello unique World"/>
  </element>
</namespace>
```

In the file use.xml we use these name spaces

use.xml:

```
<project>
  <include file="english.xml"/>
  <include file="german.xml"/>
  <unique/>      ... will prints: Hello unique World ...
  <hello/>      ... will prints: ERROR: ambiguous element "hello" ...
  <english:hello/> ... will prints: Hello World ...
  <german:hello/> ... will prints: Hallo Welt ...
</project>
```

10.3 Proposed enhancements: Scoped Includes

The input element in the Java CoG Kit workflow is scoped. This means that different scopes can be declared as follows:

```
...
<blocka>
  <include file="german.xml"/>
  <hello/> ... hello defined in german.xml ...
</blocka>
<blockb>
  <include file="english.xml"/>
  <hello/> ... hello defined in english.xml ...
</blockb>
```

However, as you can see, the scope of the definitions is not the file, but the parent of the include.

11 Lists

The next example shows the inclusion of a list, defined using the **<list>** element. The arguments to the **<list>** element are wrapped in a list, and the *hosts* variable is assigned with the value. The **<for>** element will iterate through the values of the list (but anything that can be iterated will be equally fine), and find out the date on each machine in the list in order. In other words, the date element on one host will complete execution before the evaluation will start for the next host.

```

<project>
  <include file="cogkit.xml"/>
  <element name="date" arguments="host">
    ...
  </element>
  <set name="hosts">
    <list>
      <host name="alpha.mcs.anl.gov"/>
      ...
      <host name="omega.mcs.anl.gov"/>
    </list>
  </set>
  <for name="host" in="{hosts}">
    <date host="{host}"/>
  </for>
</project>

```

Of course, the iteration can also occur in parallel:

```

...
<parallelFor name="host" in="{hosts}">
  <date host="{host}"/>
</parallelFor>
...

```

This time, dates on all hosts will be carried out in parallel.

12 Return values

In the above examples, the execution of the **<date>** element would echo a message on the console. A return could allow the caller to instead use the date information in other ways. Values are returned in Karajan whenever they are not explicitly consumed. In the above examples the return value of **<readFile>** was used to set a variable. However, if that does not happen, the value will be returned to the closest willing consumer:

```

<project>
  <include file="cogkit.xml"/>
  <element name="date" arguments="host">
    <execute executable="/bin/date" stdout="date-{host}"
             host="{host}" provider="GT2"/>
    <echo message="Job on {host} completed. Transferring output"/>
    <transfer srchost="{host}" srcfile="date-{host}"
             desthost="localhost" provider="gridftp"/>
    <echo message="Output from {host} transferred"/>

    <readFile file="date-{host}"/>
  </element>
  ...
  <parallelFor name="host" in="{hosts}">
    <set name="date">
      <date host="{host}"/>
    </set>
    <echo message="The date on {host} is {date}"/>
  </parallelFor>
  ...
</project>

```

A return value does not have to be used immediately when generated:

```

<project>
  ...
  <set name="dates">
    <list>
      <parallelFor name="host" in="{hosts}">
        <date host="{host}"/>
      </parallelFor>
    </list>
  </set>
  <echo message="The dates are {dates}"/>
  ...
</project>

```

This example illustrates that using an iterator is equivalent to statically enumerating the tasks. The above example produces the same output as the one below (minus the non-determinism associated with the concurrent aspect of the code: the order of the dates):

```

<project>
  ...
  <set name="dates">
    <list>
      <parallel>
        <date host="alpha.mcs.anl.gov"/>
        ...
        <date host="omega.mcs.anl.gov"/>
      </parallel>
    </list>
  </set>
  <echo message="The dates are {dates}"/>
  ...
</project>

```

13 Control Structures

13.1 Loops

You can include simple loops through the for and the while constructs

13.1.1 while

```
<while>
  <print message="Forever"/>
</while>
```

```
<while>
  <print message="You will see me"/>
  <condition>
    <false/>
  </condition>
  <print message="You will not see me"/>
</while>
```

13.1.2 for

13.1.3 Example

In this example we will show how to run a simple parameter study on the grid. We will use the while construct to loop through a number of parameters that are used as input to a program that is staged on a number of remote machines.

13.2 Condition

```
<if>
  <true/>
  <then>
    <print message="It's true!"/>
  </then>
  <else>
    <print message="It's all relative"/>
  </else>
</if>
```

```
<if>
  <equals value1="{a}" value2="1"/>
  <then>
    <print message="a is 1"/>
  </then>
  <equals value1="{a}" value2="2"/>
  <then>
    <print message="a is 2"/>
  </then>
  <else>
    <print message="a is {a}"/>
  </else>
</if>
```

14 Forms (Swing integration)

The following example show how easy it is to integrate forms into the Workflow. The complete example is available from CVS

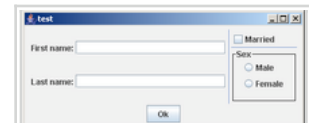


Fig 1: A Form demo

(<http://www.cogkit.org/viewcvs/viewcvs.cgi/src/cog/modules/karajan/examples/form.xml?rev=HEAD&conte>)

```

<set name="formData"
  annotation="Input lastname, firstname, marriage status, and gender">
  <form title="test" id="form" waitOn="IDOk">
    <vbox>
      <hbox>
        <vbox>
          <hbox>
            <label text="First name: "/>
            <textField id="IDFirst" columns="20"/>
          </hbox>
          <hbox>
            <label text="Last name: "/>
            <textField id="IDLast" columns="20"/>
          </hbox>
        </vbox>
        <VSeparator/>
        <vbox>
          <checkbox caption="Married" id="IDMarried" halign="0"/>
          <HSeparator/>
          <radioBox caption="Sex" id="IDSex">
            <radioButton caption="Male" id="IDMale"/>
            <radioButton caption="Female" id="IDFemale"/>
          </radioBox>
        </vbox>
      </hbox>
      <button id="IDOk" caption="Ok"/>
    </vbox>
  </form>
</set>

```

A different example includes a job submission form is included in the CVS (<http://www.cogkit.org/viewcvs/viewcvs.cgi/src/cog/modules/karajan/examples/job-submission-form.xml?rev>). It allows to send Jobs to a Grid. Naturally you should be adapting this code according to your available infrastructure.

15 System Command

System commands such as the start of shell scripts are easily doable through task executions. Assume you have a script in /path/to/script, you can execute it as follows:

```

task:execute(executable="/bin/sh",
             args="/path/to/script",
             provider="local",
             redirect=true/false)

```

16 Invoking Java methods with the Java Library

http://www.cogkit.org/viewcvs/viewcvs.cgi/*checkout*/src/cog/modules/karajan/examples/stopwatch.xml?rev

17 Other Available Features

A Section listing other available features will be included her. These features are not explained in detail. We will refer to the manual for a complete description.

18 Limitations

For the academics

- write about the GT limitations
- write about the SSH limitations
- we are only as powerful as the underlying middleware
- How many parallel statements can be handled?

The known issues file explains the current limitations. For more than approximately 40000 parallel threads, a deadlock of the system can occur. This will be solved in the next version.

- what is the relationship between parallel blocks and memory.

It depends. No actual OS threads are used. Whenever a Karajan thread is created, it is attached a stack that is used to hold the state of that thread. Existing frames that belong to callers are shallow-copied. New frames are created independently. Depending on the amount of variables, and the depth of invocation, more or less memory is used. However, simply creating a Karajan thread costs somewhere on the scale of less than 1KB.

- What is the relationship between variables and their memory consumption?

Variables are stored in frames which are hashtables. So a variable will consume the amount of memory needed for the corresponding Java object, the amount of memory needed for the name of the variable, and the amount of memory needed to create the entry in the hashtable.

19 Appendix

Java CoG Kit Workflow Guide: Proposed Extensions

20 References

- Gregor von Laszewski, A Loosely Coupled Metacomputer: Cooperating Job Submissions Across Multiple Supercomputing Sites, *Concurrency, Experience, and Practice*, Dec. 1999, Vol. 11, No. 5, pages 933-948, The initial version of this paper was available in 1996, <http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--CooperatingJobs.ps>
- The Java CoG Kit Web page at <http://www.cogkit.org>
- Gregor von Laszewski and Mike Hategan, *Grid Workflow - An Integrated Approach*, Argonne National Laboratory, Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60440, <http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-workflow-draft.pdf>
- Gregor von Laszewski and Ian Foster and Jarek Gawor and Peter Lane, *A Java Commodity Grid Kit, Concurrency and Computation: Practice and Experience*, 2001, Vol. 13, No. 8-9, pages 643-662, <http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--cog-cpe-final.pdf>

21 Current Tasks

■ Gaussian example to be tried with the workflow service (Deepti)

Testing of gaussian on the chemistry theory cluster with help from Mike or get info on workflow service docs. Workflow Gaussian Example

Retrieved from "http://wiki.cogkit.org/index.php/Java_CoG_Kit_Workflow_Guide"

Category: Workflow

- This page was last modified 04:26, 21 August 2006.
- This page has been accessed 1,027 times.
- [Privacy policy](#)
- [About Java CoG Kit](#)
- [Disclaimers](#)