OpenKnowledge Groovy
OKC Approach

04 Nov 2008

Table of Contents

Overview

Groovy is an agile and dynamic language for the Java Virtual Machine, it builds upon the strengths of Java but has additional power features inspired by languages like Python, Ruby and other scripting languages. Groovy also increases developer productivity by reducing scaffolding code. It integrates with all existing Java objects and libraries seamlessly, therefore Groovy can be used to create OKCs and running within the OK kernel. Moreover Java developers are able to use Groovy with almost-zero learning curve, and it's also easy for non programmers to learn and use, so non programmers could even write OKCs themselves.

This document shows how to develop a simple OKC in both Groovy and Java for comparison. It also demonstrates more complicated bio-infomatic case studies using the Open Knowledge framework with MSMS and Yeast, as well as the performance test results for the case studies.

Simple Groovy Sample

This simple Groovy OKC approach sample code shows a basic interaction between two peers for comparison we have given the code for both Java and Groovy implementations.

LCC

LCC code for the sample:

```
r( requester, initial )
r( service, necessary, 1 )

a(requester, A) ::
   request(Input) => a(service, B) <- query(Input)
  then
  response(Result) <= a(service, B) <- printResults(Result)

a(service, B) ::
   request(Input) <= a(requester, A)
   then
   response(Result) => a(requester, A) <- process(Input, Result )
```

Groovy Implementation

TestRequesterRole

```
package testGroovyOKC

import org.openk.core.OKC.impl.OKCFacadeImpl;
import org.openk.core.module.interpreter.Argument;

class TestRequesterRole extends OKCFacadeImpl {
	boolean query(Argument input){
		input.value = "Test Input Values"
		println "TestRequesterRole::query input=["+input.value+"]"
		true
	}
	boolean printResults(Argument result){
		println "TestRequesterRole::printresults  result=["+result.value+"]"
		groovy.swing.SwingBuilder builder = new groovy.swing.SwingBuilder()
		builder.optionPane(message: result.value).createDialog(null, 'Test
Message').show()
		true
	}
}
```

TestServiceRole

```
package testGroovyOKC

import org.openk.core.OKC.impl.OKCFacadeImpl;
import org.openk.core.module.interpreter.Argument;

class TestServiceRole extends OKCFacadeImpl {
	boolean process(Argument input, Argument result){
		println "TestServiceRole::process input=["+input.value+"]"
		result.value = "Processed ----"+input.value;
		true
	}
```

```
}
```

Java Implementation

TestRequesterRole

```
package testJavaOKC;

import org.openk.core.OKC.impl.OKCFacadeImpl;
import org.openk.core.module.interpreter.Argument;

public class TestRequesterRole extends OKCFacadeImpl {
	public boolean query(Argument input){
		input.setValue("Test Input Values");

		System.out.println("TestRequesterRole::query input=["+input.getValue()+"]");
		return true;
	}

	public boolean printResults(Argument result){
		System.out.println( "TestRequesterRole::printresults
result=["+result.getValue()+"]" );

		javax.swing.JOptionPane.showMessageDialog(null, result.getValue().toString(),
"Test Message", javax.swing.JOptionPane.INFORMATION_MESSAGE);
		return true;
	}
}
```

TestServiceRole

```
package testJavaOKC;

import org.openk.core.OKC.impl.OKCFacadeImpl;
import org.openk.core.module.interpreter.Argument;

public class TestServiceRole extends OKCFacadeImpl {
	public boolean process(Argument input, Argument result){
		System.out.println( "TestServiceRole::process input=["+input.getValue()+"]");

		result.setValue("Processed ----"+input.getValue() );
		return true;
	}
}
```

Java Bioinfo

This is the Java implementation of Bioinfo project, it's SVN repository location is at:
http://fountain.ecs.soton.ac.uk/ok/repos/protocols/bioinfo-ed/

MSMS Case Study

This case study can only run on a Windows machine, as the program and scripts involved only works in Windows. The entry point for running MSMS test case is *org.openk.okc.bioinfo.msms.MSMSExp*

The OKCs used in this test case are:

org.openk.okc.bioinfo.okc.ExperimenterOKC
org.openk.okc.bioinfo.okc.CoordinatorOKC
org.openk.okc.bioinfo.okc.PEPNOVODataSourceOKC
org.openk.okc.bioinfo.okc.LUTEFISKDataSourceOKC
org.openk.okc.bioinfo.okc.PEAKSDataSourceOKC
org.openk.okc.bioinfo.okc.MSMSComparerOKC

Yeast Case Study

This case study can only run on a Linux machine, as the program and scripts involved only work in Linux. The entry point for running Yeast test case is *org.openk.okc.bioinfo.yeast.YeastExpBasic*.

The OKCs used in this test case are:

org.openk.okc.bioinfo.okc.ExperimenterOKC
org.openk.okc.bioinfo.okc.CoordinatorOKC
org.openk.okc.bioinfo.okc.MODBASESourceOKC
org.openk.okc.bioinfo.okc.SAMSourceOKC
org.openk.okc.bioinfo.okc.SWISSSourceOKC
org.openk.okc.bioinfo.okc.DataComparerOKC

Performance Test Monitor

To see how much of the machines resources the experiment if using a Performance Test Monitor is built into the Bioinfo project, which can be used to monitor the CPU and Memory usage, whilst the test case is running. Both Windows and Linux performance monitors are implemented.

Windows performance monitor is located at: *org.openk.okc.bioinfo.util.CPUmon*
Linux performance monitor is located at: *org.openk.okc.bioinfo.util.CPUmonLinux*

Configure Performance Monitor

The output settings for the performance monitor can be configured using the variables *linuxMonOutPath* and *winMonOutPath* at the top of *org.openk.okc.bioinfo.okc.ExperimenterOKC* class.

Running the Performance Monitor

To use the performance monitor, just switch the *startCPUMon* flag on the top of *org.openk.okc.bioinfo.okc.ExperimenterOKC* to “true”.

For MSMS, you will also need to switch *cpuMon* flag on the top of *org.openk.okc.bioinfo.okc.MSMSComparerOKC* to “true”.

For Yeast, you will also need to switch *cpuMon* flag on the top of *org.openk.okc.bioinfo.okc.DataComparerOKC* to “true”.

Performance Analysis Program

Once you have got all the performance results you needed by running the experiments many times whilst the Performance Test Monitor turned on, you can use the Performance Analysis Program to generate the overall performance report. The program automatically runs through all the performance result and calculates the average for each experiments.

The program is located at *org.openk.okc.bioinfo.util.PerformanceAnalysis*

You will need to give the program a directory where performance results located, this directory path can be given by using either program arguments or *pDataPath* variable at the top of the *PerformanceAnalysis* file.

Groovy Bioinfo

This is the Groovy implementation of Bioinfo project, it's SVN repository location is at:
http://fountain.ecs.soton.ac.uk/ok/repos/protocols/bioinfoGroovy/

MSMS Case Study

This case study can only run on a Windows machine, as the program and scripts involved only work in Windows. The entry point for running Groovy MSMS test case is *com.gulfstream.software.bioinfo.groovy.msms.MSMSExp*

The OKCs used in this test case are:

com.gulfstream.software.bioinfo.groovy.okc.ExperimenterOKC
com.gulfstream.software.bioinfo.groovy.okc.CoordinatorOKC
com.gulfstream.software.bioinfo.groovy.msms.okc.PEPNOVODataSourceOKC
com.gulfstream.software.bioinfo.groovy.msms.okc.LUTEFISKDataSourceOKC

com.gulfstream.software.bioinfo.groovy.msms.okc.PEAKSDataSourceOKC
com.gulfstream.software.bioinfo.groovy.msms.okc.MSMSComparerOKC

Yeast Case Study

This case study can only run on a Linux machine, as the program and scripts involved only work in Linux. The entry point for running Groovy Yeast test case is *com.gulfstream.software.bioinfo.groovy.yeast.YeastExpBasic.*

The OKCs used in this test case are:

com.gulfstream.software.bioinfo.groovy.okc.ExperimenterOKC
com.gulfstream.software.bioinfo.groovy.okc.CoordinatorOKC
com.gulfstream.software.bioinfo.groovy.yeast.MODBASESourceOKC
com.gulfstream.software.bioinfo.groovy.yeast.SAMSourceOKC
com.gulfstream.software.bioinfo.groovy.yeast.SWISSSourceOKC
com.gulfstream.software.bioinfo.groovy.yeast.DataComparerOKC

Performance Test Monitor

The same performance test monitor is used for this case study, refer to *Performance Test Monitor* for instructions.

Configure Performance Monitor

The output settings for the performance monitor can be configured using the variables *linuxMonOutPath* and *winMonOutPath* at the top of the *com.gulfstream.software.bioinfo.groovy.okc.ExperimenterOKC* class.

Running the Performance Monitor

To use the performance monitor, just switch the *startCPUMon* flag on the top of *com.gulfstream.software.bioinfo.groovy.okc.ExperimenterOKC* to “true”.

For MSMS, you will also need to switch *cpuMon* flag on the top of *com.gulfstream.software.bioinfo.groovy.msms.okc.MSMSComparerOKC* to “true”.

For Yeast, you will also need to switch *cpuMon* flag on the top of *com.gulfstream.software.bioinfo.groovy.yeast.DataComparerOKC* to “true”.

Performance Analysis Program

The same performance analysis program can be used in Groovy versions of the implementations, refer to *Performance Analysis Program* for more information.

Performance Test Results

Each test case was run 6 times for performance test results, the results are kept in Groovy Bioinfo project in the *performance* directory.

Note: Windows and Linux are using different mechanism for calculation the CPU usage, hence it is not comparable between Windows and Linux CPU usage. However Java and Groovy test cases running on the same operating system are comparable.

MSMS performance results

Machine specification for this test case.

System:
Microsoft Windows XP Professional Version 2002 Service Pack 2
Computer:
Intel(R) Core(TM)2 CPU
T5200 @ 1.60 GHz
2 GB of RAM

Here are the results calculated from Performance Analysis Program:

```
File:groovyUsage1.txt, Total Time:15281, Avg CPU:6.96, Avg Memory:60543803, Max CPU:53, Max
Memeory:70662312
File:groovyUsage2.txt, Total Time:15000, Avg CPU:7.913, Avg Memory:61678652, Max CPU:54.6,
Max Memeory:73730176
File:groovyUsage3.txt, Total Time:15531, Avg CPU:8.668, Avg Memory:60346408, Max CPU:57.8,
Max Memeory:76108752
File:groovyUsage4.txt, Total Time:14516, Avg CPU:7.331, Avg Memory:60727143, Max CPU:56.2,
Max Memeory:64307736
File:groovyUsage5.txt, Total Time:15187, Avg CPU:7.547, Avg Memory:61289380, Max CPU:61,
Max Memeory:67580104
File:groovyUsage6.txt, Total Time:15438, Avg CPU:7.093, Avg Memory:60810511, Max CPU:50,
Max Memeory:73126584
File:javaUsage1.txt, Total Time:14594, Avg CPU:5.459, Avg Memory:49335602, Max CPU:54.6,
Max Memeory:70355016
File:javaUsage2.txt, Total Time:14781, Avg CPU:5.566, Avg Memory:49433756, Max CPU:57.8,
Max Memeory:59584560
File:javaUsage3.txt, Total Time:14062, Avg CPU:5.446, Avg Memory:48946572, Max CPU:54.8,
Max Memeory:57436856
File:javaUsage4.txt, Total Time:14875, Avg CPU:4.824, Avg Memory:50551763, Max CPU:42.2,
Max Memeory:71386128
File:javaUsage5.txt, Total Time:14016, Avg CPU:5.407, Avg Memory:49708503, Max CPU:53, Max
Memeory:68115512
File:javaUsage6.txt, Total Time:14046, Avg CPU:5.521, Avg Memory:50107555, Max CPU:53, Max
Memeory:62134704
Groovy Usage: Avg Time:15158, Avg CPU:7.585, Avg Memory:60899316
Java Usage: Avg Time:14395, Avg CPU:5.37, Avg Memory:49680625
```

Yeast performance results

Machine specification for this test case.

System:
Linux Fedora 8, 2.6.23.1-42.fc8
Computer:
Intel(R) Core(TM)2 CPU
T5200 @ 1.60 GHz
2 GB of RAM

Here is the results calculated from Performance Analysis Program:

```
File:groovyUsage1.txt, Total Time:14321, Avg CPU:86.043, Avg Memory:64540306, Max CPU:162,
Max Memeory:95676760
File:groovyUsage2.txt, Total Time:13547, Avg CPU:85.696, Avg Memory:64576479, Max CPU:144,
Max Memeory:92343912
File:groovyUsage3.txt, Total Time:14658, Avg CPU:85.739, Avg Memory:68759438, Max CPU:144,
Max Memeory:93957560
File:groovyUsage4.txt, Total Time:13995, Avg CPU:86.348, Avg Memory:65883130, Max CPU:158,
Max Memeory:96695632
File:groovyUsage5.txt, Total Time:13970, Avg CPU:85.957, Avg Memory:69230157, Max CPU:146,
Max Memeory:95836288
File:groovyUsage6.txt, Total Time:14691, Avg CPU:84.958, Avg Memory:72602975, Max CPU:163,
Max Memeory:97778528
File:javaUsage1.txt, Total Time:10220, Avg CPU:60.111, Avg Memory:63962163, Max CPU:122,
Max Memeory:94104632
File:javaUsage2.txt, Total Time:12027, Avg CPU:57.722, Avg Memory:65651809, Max CPU:116,
Max Memeory:89919808
File:javaUsage3.txt, Total Time:11277, Avg CPU:57.556, Avg Memory:68844332, Max CPU:126,
Max Memeory:93261440
File:javaUsage4.txt, Total Time:10085, Avg CPU:59, Avg Memory:69808264, Max CPU:120, Max
Memeory:85827272
File:javaUsage5.txt, Total Time:10841, Avg CPU:60.444, Avg Memory:62280783, Max CPU:130,
Max Memeory:87223944
File:javaUsage6.txt, Total Time:10760, Avg CPU:56.647, Avg Memory:62897021, Max CPU:116,
Max Memeory:93312496
Groovy Usage: Avg Time:14197, Avg CPU:85.79, Avg Memory:67598747
Java Usage: Avg Time:10868, Avg CPU:58.58, Avg Memory:65574062
```