CMU	17-654	&	17-754
CIVIO	1/ 00-	æ	1/ /34

Analysis of Software Artifacts Individual Project: Tool Analysis Spring 2006

May 18, 2006 Eun-young Cho echo1@andrew.cmu.edu

# JProfiler: Code Coverage Analysis Tool for OMP Project

# Table of Contents

1. Project Objectives	2
2. Background	
2.1 About JProfiler:	
2.2 About Nemo and JProfiler's scope on Nemo	
3. Experimental Setup	
3.1 JProfiler Installation and Setup	7
3.2. Nemo Installation	
3.3. Nemo Execution	9
4. Analysis of Results	9
4.1 Memory View through Nemo	9
4.2 CPU View through Nemo	
4.3 Thread View through Nemo	15
4.4 VMtelemetry View through Nemo	
5. Lessons Learned.	
5.1 General Characteristics of JProfiler	
5.2 Benefits to the MSE studio project	19
5.3 Drawbacks	
6. Conclusions	21
7. References	21

# 1. Project Objectives

To use JProfiler, an analysis tool, to report performance losses to:

- Report memory leaks on Nemo (an example of an Overlay Multicast Protocol, OMP)
- Resolve threading issues on Nemo
- Gain insight into the group management aspect of OMP using Nemo
- Use the gained knowledge for the MSE POSDATA studio project

# 2. Background

## 2.1 About JProfiler:

JProfiler is a unique tool when compared to any of its peers as:

- The tool uses a combined approach to provide different perspectives.
- The tool provides a faster 4 in 1 approach where the 4 views in one window correspond to Memory views, CPU Views, Thread Views and VM Telemetry Views.

The details of the aforementioned views are given below:

# Memory Views

This view provides for:

- Heap walker styled drill down showing object references
- The drill down reports problem spots with a tree like representation of the Heap data structure.
- Detailed browsing of the Heap structure, in order to get information on memory and object references.

## **CPU Views**

This view provides for:

- Showing threads information on invocation of threads and their back traces.
- Filtering mechanism enables the customizing the data at hand to one's own perspective
- A real time dynamic picture on the views

## **Thread Views**

This view provides for:

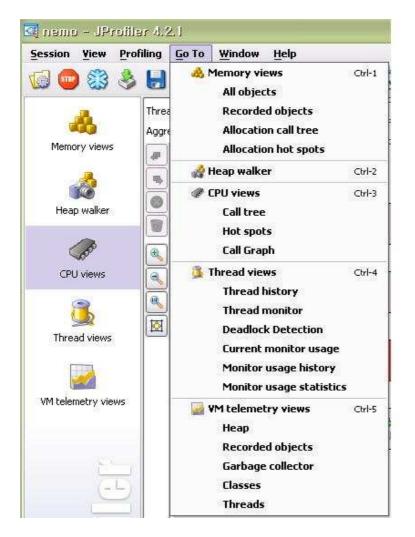
- Deadlock profiling by showing thread monitoring and colored coded thread history which enables programmers to catch deadlocks where they might potentially exist.
- The thread debugger is also included in the JProfiler's installation package.

## **VM Telemetry Views**

This view provides for:

- Information on the Virtual machine's parameter from the moment the JVM starts.
- Each view breaks down into a sub view to enhance display and readability. This also makes the switching between views very easy and vastly possible.

The following snapshot shows JProfiler with its major views drop-down menu.



Followings are the comparison with similar tools in 2003. It's a competitive tool in perspective of features and costs. JProfiler got 2003 and 2005 Java Developer's Journal Readers' Choice Awards as the Best Java Profiling/Testing Tool. JProfiler was nominated and adjudged as the best Java Profiling tool for the 2003 and 2005 Java Developer's Journal Readers' Choice Award.

	Optimizeit Suite	JProbe Suite	JProfiler
Version	5.0	5.0	2.2.1
Price	\$1,599	\$2,000 <sup>1</sup>	\$499
Free evaluation	Yes	Yes	Yes
Online (built-in) help	Yes	Yes (JavaHelp)	Yes (JavaHelp) <sup>2</sup>
Is help context- sensitive?	Yes	Yes	Yes
Built-in tutorials	Yes	Yes	No <sup>3</sup>
Paper documentation	No	Yes	No
Number of tool modules	3 (Profiler, Thread Debugger, Code Coverage)	4 (Profiler, Coverage, Memory Debugger, Threadalyzer)	0 (all-in-one)
Tool modules sold separately?	No	Yes	No
CPU profiler	Yes (not real time)	Yes (not real time)	Yes (real time)
Object/heap profiler	Yes	Yes	Yes
Thread profiler	Yes	Yes	Yes
Deadlock detection	Automated and visual	Automated	Manual
Race condition detection	No	Yes	No
Code coverage	Yes	Yes	No
Multi-JVM support	Yes	Yes	Yes
Drill-down to source	Yes	Yes	Yes
Drill-down to bytecode	No	Yes	Yes
Remote profiling*	Yes	Yes	Yes
Automated profiling**	Yes	Yes	Yes
IDE integration	Yes	Yes	Yes
Report generation	Yes	Yes	Yes
Host platform licensing policy	Multiplatform and single-platform licenses	Single platform	Multiplatform
Website	www.borland.com/optimizeit	www.jprobe.com	www.jprofiler.com
Ease of use	7/10	4/10	8/10

Source: http://www.javaworld.com/javaworld/jw-08-2003/jw-0822-profiler.html

\* Remote profiling: The ability to profile a Java program executing on a machine other than your development machine

\*\* Automated profiling: The ability to perform unattended overnight profiling sessions; in other words, command-line-driven operation with no GUI

<sup>1</sup> JProbe Suite price includes one year of Gold Support (technical support)

<sup>2</sup> ej-technologies' JProfiler Online Help contains almost no screenshots of views or dialogs

<sup>3</sup> ej-technologies' lack of explicit tutorials is partly compensated by some demo sessions

This is particularly of interest because on POSDATA, SI Company in Korea, my studio project requires the use of OMP in order to broadcast video stream to particular nodes through the use of group management. Though the focus of the JProfiler testing is not for OMP in general, the program of interest is Nemo, a multicast protocol that uses group management. Here Nemo is an existing OMP project that serves a good example to show the kind of problems and parameters one may have to face or think about when one is realizing an OMP. The main types of nodes are:

- **Bootstrap Node**: Nodes that serve as the leader of a cluster or a segment of a group. They serve as the meeting point for publishers and subscribers
- **Publisher Nodes**: Publish data and send data to subscribers. They may also be known as co-leaders in the OMP terminology. The co-leader is like a server for a particular layer consisting of OMP clients.
- Subscriber Nodes: Receive data from publishers. They can be referred to as clients.

Followings are the script in case of running for each node role.

🙀 nemo	[3] - JProfiler 4.2.1	
Terminal	Window	1000
	irectory: D:#1MSE#6Posdata#developmentManagerActionTask#0MPjava# call: C:#Java#jdk#bin#javaw,exe −agentlib:jprofilerti=port=31782	
JProfiler>	> Protocol version 22	
JProfiler>	> Using JVMTI	
JProfiler>	> 32-bit library	
JProfiler>	> Listening on port: 31782,	
JProfiler>	> Native library initialized	
JProfiler>	> Waiting for a connection from the JProfiler GUI ,,,	
JProfiler>	> Using dynamic instrumentation	
JProfiler>	> Time measurement: elapsed time	
JProfiler>	> CPU profiling enabled	
	> Hotspot compiler enabled	
JProf i ler>	Starting edu/nwu/nemo/examples/MulticastAgent ,,,	
setup boot:	tstrap service	
Service mu:	ust implement interface edu,nwu,multicast,api,lMulticastSubscrit	per and cor
Type 'quit	t' to terminate the session	
5 - 128 - 1.4 C		
•		•
Input:		

(a) Bootstrap node

🖬 nemo	[3] - JProiller 4.2.1	
Terminal	<u>W</u> indow	
	rectory: D:#1MSE₩6Posdata₩developmentManagerActionTa all: C:#Java#jdk₩bin₩javaw,exe -agentlib:jprofilerti	
JProfiler>	Protocol version 22	
	Using JVMTI	
	32-bit library	
JProfiler>	Listening on port: 31777,	
	Native library initialized	
JProfiler>	Waiting for a connection from the JProfiler GUI	
JProfiler>	Using dynamic instrumentation	
JProfiler>	Time measurement: elapsed time	
JProfiler>	CPU profiling enabled	
JProfiler>	Hotspot compiler enabled	
JProfiler>	Starting edu/nwu/nemo/examples/MulticastAgent ,,,	
setup mult	icast service	
Publishing	at 10 ms intervals	
Type 'quit	' to terminate the session	
log4j:WARN	No appenders could be found for logger (edu, nwu, ner	mo,bll,AgentTasks);
log4j:WARN	Please initialize the log4j system properly,	
JProfiler>	Disconnected,	
•		
Input:		

(b) Publisher Node

a nemo		
Terminal	Window	
and the second second	rectory: D:#1MSE₩6Posdata₩developmentManagerActionTas all: C:#Java₩jdk₩bin₩javaw,exe —agentlib:jprofilerti=	
JProfiler>	Protocol version 22	
JProfiler>	Using JVMTI	
JProfiler>	32-bit library	
JProfiler>	Listening on port: 31776,	
JProfiler>	Native library initialized	
JProfiler>	Waiting for a connection from the JProfiler GUI	
JProfiler>	Using dynamic instrumentation	
JProfiler>	Time measurement: elapsed time	
JProfiler>	CPU profiling enabled	
JProfiler>	Hotspot compiler enabled	
JProfiler>	Starting edu/nwu/nemo/examples/MulticastAgent	
setup boots	strap service	
Subscribing	g to multicast session	
Type 'quit	' to terminate the session	
Service mus	st implement interface edu,nwu,multicast,api,lMultica	stSubscriber and co
JProfiler>	Disconnected,	
•		Þ
Input:		

(c) Subscriber Node

Figure 1: Execution script using the JProfiler tool

### 2.2 About Nemo and JProfiler's scope on Nemo

Nemo, an open source project, implements the concept of Overlay Multicast Protocol which is a networking protocol to share a single data stream between a large number of connecting clients without degradation of the performance or increase in network cost. In the present situation on the MSE project, there is a requirement to deal with the high degree of variability that may exist in the network on account of the join and leave operations of the clients. This variability arises from the dynamic situation of a large number of nodes joining and leaving, the network. The aim of any multicast protocol is to achieve this variability without giving up on end to end delays and providing for an additional network costs.

# 3. Experimental Setup

### 3.1 JProfiler Installation and Setup

In order to run JProfiler, several steps should be done. First, download the evaluation version (JProfiler 4.2.1) of JProfiler which is jprofiler\_windows\_4\_2\_1.exe in the website, <u>http://www.ej-technologies.com/download/jprofiler/trial.php</u>. Second, install JProfiler. Finally, run JProfiler with personal evaluation key which received via email. For windows user below environment setup is needed.

Session name : nemo	Id: 1.
Session type	· · · · · · · · · · · · · · · · · · ·
Local settings	
Java VM:	Sun 1.5.0_06
Working directory:	pmentManagerActionTask\OMPjava\nemo-1.0-sources\edu\nwu\nemo
VM arguments:	
Main class or executable	JAR: Ledu.nwu.nemo.examples.MulticastAgent
Arguments:	80 128.237.234.114:80 10
🔲 open browser with U	RL:
<ul> <li>class path</li> <li>source path</li> <li>native library path</li> </ul>	:a\developmentManagerActionTask\OMPjava\Jars\colt.jar :a\developmentManagerActionTask\OMPjava\Jars\commons-collections-3.1.jar :a\developmentManagerActionTask\OMPjava\Jars\commons-logging-1.0.3.jar :a\developmentManagerActionTask\OMPjava\Jars\comcurrent.jar :a\developmentManagerActionTask\OMPjava\Jars\concurrent.jar :a\developmentManagerActionTask\OMPjava\Jarsbindall-1.0-sources.jar :a\developmentManagerActionTask\OMPjava\Jarsbindall-1.0-sources.jar :a\developmentManagerActionTask\OMPjava\Jarsbindall-1.0-sources.jar :a\developmentManagerActionTask\OMPjava\Jarsbindall-1.0-sources.jar :a\developmentManagerActionTask\OMPjava\Jarsbindall-1.0.jar
	:a\developmentManagerActionTask\OMPjava\Jars\nemo-1.0-sources.jar :a\developmentManagerActionTask\OMPjava\Jars\nucommon-1.0-sources.jar

Java VMs	Filter sets	Default java file path	IDE integrations	Miscellaneous	
		🔘 class path 🏾 🔘 sourc	e path 🔘 native l	ibrary path	
- 8 A & B & B & B & B & B & B & B & B & B &	8423 (CC 2016) (CC 2016)	evelopmentManagerAction	승규는 것이 같은 것이 없는 것이 없다.		
🗿 D:\1MS	E\6Posdata\de	evelopmentManagerAction evelopmentManagerAction evelopmentManagerAction	Task\OMPjava\nucc		
	E\6Posdata\de	evelopmentManagerAction	Task\OMPjava\nucc	mmon-1.0-sources	

After successful setting of the target programs, a program can be monitored by JProfiler in four areas.

🔯 nemo - JProiil	er 4.2.1
Session View Pro	filing <u>G</u> o To <u>Wi</u> ndow <u>H</u> elp
🕼 🕨 🌒 🕹	🖩 🤌 🗘 🖍 i 🖏 🐌 i 🎲 🗔 i 🕑 i 🎘 🛸
	Aggregation level: Classes
Memory views	Name
Heap walker	
55	Start Program and Profile it [F11]
CPU views	
3	
Thread views	
VM telemetry views	

### 3.2. Nemo Installation

Nemo is an open-source overlay multicast protocol for streaming applications provided by Northwestern University. In order to run Nemo, Nemo source file and additional jar files are need to Download. The source website is as follows.

<u>http://www.aqualab.cs.northwestern.edu/projects/nemo/download.php</u> or <u>http://sourceforge.net/project/showfiles.php?group\_id=160473</u>

onet				Lo	og In - Create Account	Q overlay	multicast S	earch
SF.net Proj	ects My Pag	je Help						
Software Map Crea	te Project New Rel	eases Top Projects New	/ Projects 🚥	Help Wanted				
SF.net » Projects » Nem	no - Resilient Overlay Mu	ticast » Files					6	
Nemo - Resilio	ent Overlay M	ulticast				🖉 Stats -	Activity: 81.03%	RSS
CVS   Files	silient Overlay Mu	and the second second				From These Pre		
							rackspace	
achieves high deli	very ratio without sa	multicast protocol for strea crificing end-to-end latency			Ads by Google		rackspace	
	very ratio without sa				<b>^</b>		rackspace	
achieves high deli	very ratio without sa				Ads by Geogle Domain N	ames only		
achieves high deli	very ratio without sa	crificing end-to-end latency	or incurring ad Notes /	Iditional costs.	Ads by Google Domain N Free Page,	*	& Privacy	
achieves high deli Latest File Relea Package	very ratio without sa ses Release	rificing end-to-end latency Date	or incurring ad Notes / Monitor	lditional costs.	Ads by Google Domain N Free Page,	ames only DNS Control	& Privacy	
achieves high deli Latest File Relea Package heimdall	very ratio without sa ses Release 1.0	Date February 24, 2006	or incurring ad Notes / Monitor @ - 63	lditional costs. Downloads Download	Ad: by Google Domain N Free Page, Move your s	ames only DNS Control	& Privacy	

### 3.3. Nemo Execution

Nemo provides sample program named MulticastAgent.java to test overlay multicast protocol. The program can be run using three different set of parameters. When it runs using one parameter which is port number, the agent program runs as a bootstrap. A subscriber needs one more parameter, the address of bootstrap agent. A publisher needs additional packet sending interval. In this experiment, I start using parameters in the MulticastAgent.java as follows.

Node type	Input Parameter	Meaning
Bootstrap	80	Local Port Number
Subscriber	80 128.237.234.114:80	Port No., Bootstrap IP + Port No.
Publisher	80 128.237.234.114:80 10	Port No., Bootstrap IP + Port No., Publish-Rate

## 4. Analysis of Results

### 4.1 Memory View through Nemo

As mentioned before, several memory views support heap analysis. Following snapshot shows the class monitor subview.

As can be seen through the memory view, the initial large allocation is for the logging feature provided inbuilt into Nemo. This feature is commonly found on all the three node types. The next large memory allocations are for the receive queue and the packet socket. When compared at the method aggregation level, the Nemo BootstrapService.setup is the class with maximum allocation to its methods.

In following memory views, initial large memory allocation is for the log. It's common on three node types. And next common allocations are for the ReceiveQueue and PacketSocket.

In method aggregation level, NemoBootstrapService.setup is major allocated class and its method.

For the bootstrap mode the view is defined below:

nemo [9] - JP			
ession <u>Y</u> iew <u>P</u> ro	filing <u>G</u> o To <u>W</u> indow <u>H</u> elp		
🏼 📟 😂 🗞			
	Aggregation level: Classes		[
<b>400</b>	Name	Instance count 🔺	Size
Memory views	char[]	4,648	682 kB
and a state of the	java.lang.String	3,021	72,504 bytes
	int[]	1,757	105 kE
	short[]	<b>1,273</b>	75,848 bytes
	<class>[]</class>	1,064	53,072 bytes
Heap walker	byte[]	1,040	318 kE
	java.lang.Class	941	334 kE
B	java.util.TreeMap\$Entry	771	24,672 bytes
- St	java.util.HashMap\$Entry	488	11,712 bytes
CPU views	java.util.Hashtable\$Entry	326	7,824 bytes
	java.util.LinkedHashMap\$Entry	266	8,512 byte:
9	java.util.LinkedList\$Entry	211	5,064 byte:
5	iava.lang.reflect.Method	151	12,080 bytes
Thread views	java.lang.reflect.Constructor	101	6,464 bytes
micod views	com.jprofiler.agent.L\$_B	96	1,536 bytes
	iava.net.URL	89	4,984 bytes
100	java.util.Date	86	2,064 byte:
(maine)	java.util.HashMap	84	3,360 byte:
VM telemetry views	java.lang.Integer	77	1,232 byte
	java.lang.reflect.Field	77	5,544 byte
	java.io.ExpiringCache\$Entry	76	1,824 byte
	java.util.LinkedList	75	1,800 byte:
183- I	java.lang.ref.SoftReference	73	2,336 byte
	java.lang.ref.Finalizer	63	2,016 bytes
( - L- )	java.io.FileDescriptor	63	1,512 bytes
	java.lang.Object	61	488 bytes
	java.io.FileInputStream	57	912 bytes
	java.lang.StringBuilder	53	848 bytes
	edu.nwu.net.dll.SimpleField	51	1,224 bytes
2	org.apache.log4j.CategoryKey	46	736 bytes
	lines on a sile. Den side of Constant of	L AT.	1.000 hits
	View Filters:		Reset view filters
	All objects Recorded objects Allocat	ion call tree Allocation hot spots	

🔝 🧰 🏂 🔈	📙 🕭 🚳 🟠	- 🎎 🧇 - 🛞 🯹 - 🞯 - 🗐 -	<b>I</b>			
	Recorded allocations of: Liveness mode:	All classes Live objects		tat exercit		
Meniory views	Aggregation level:	Methods	<b>•</b>	Filtered classes: Allocated memory		y L
Heap walker CPU views Thread views WM telemetry views	Image: Constraint of the second sec	mons.loggingLtogFactory.getLog(java.lang.class) > 536 (8 - 4), 72 allor, edu.mwu.neno.examples. Multicast 14 bytes - 36 allor, edu.mwu.net.bll.ReceiveQueux.cclint 18 bytes - 36 allor, edu.mwu.net.bll.ReceiveQueux.cclint 18 bytes - 9 alloc, edu.mwu.net.bll.ReceiveQueux.cclint 19 bytes - 9 alloc, edu.mwu.net.bll.ClasnableManger.clint 19 bytes - 7 alloc, edu.mwu.net.bll.ClasnableManger.clint 19 bytes - 7 alloc, edu.mwu.net.bll.ClasnableManger.clint 19 bytes - 7 alloc, edu.mwu.net.bll.RelableComnectionMappi 19 bytes - 7 alloc, edu.mwu.met.bll.NetableStableXtockto.clint 19 bytes - 7 alloc, edu.mwu.met.bll.NetableXtocktorManger, 19 bytes - 7 alloc, edu.mwu.met.bll.NetableXtocktorManger, 19 bytes - 7 alloc, edu.mwu.met.bll.NetableXtocktor.clint 19 bytes - 7 alloc, edu.mwu.met.bll.StreemcYcotManger, 19 bytes - 7 alloc, edu.mwu.met.bll.StreemcYcotAthanger, 19 byte	> ' - ' - ' - ' - ' - ' - ' - ' - ' - ' -		kB (57 %)	4,823
	📙 💩 🛅 🛯 🖓 - 216	bytes - 7 alloc, edu pwu net dli DatagramPacket «clinit»				

For the publisher node the view is defined below:

nemo [3] - JPr			
ession <u>Y</u> iew <u>P</u> rol	iling <u>G</u> o To <u>W</u> indow <u>H</u> elp		
ځ 😢 🚥 😫	📙 🔌 🚳 i 🎎 💝 i 🎡 🗖 i 🎯 i 📑 i	🏽 👏	
*	Recorded allocations of: All classes Liveness mode: Live objects		
Memory views	Aggregation level: Methods	Filtered classes: show separate	ly [·
-	Hot spot	Allocated memory 🖌	Allocations
1	Org.apache.commons.logging.LogFactory.getLog(java.lang.Class)	553 kB (57 %)	4,823
	💁 🚹 edu.nwu.net.bll.DatagramSocket\$Receiver. <init></init>	65,552 bytes (6 %)	1
Heap walker	💁 🚠 java.util.LinkedHashSet. <init>()</init>	34,144 bytes (3 %)	776
	🔶 🥂 java.nio.channels.Selector.open	22,680 bytes (2 %)	264
-	📴 💁 cern.jet.random.engine.MersenneTwister. <init></init>	20,096 bytes (2 %)	8
6	🖗 🔶 java.lang.StringBuffer.append(java.lang.Object)	19,320 bytes (1 %)	397
-167	👳 🕭 java.lang.reflect.Constructor.newInstance	17,376 bytes (1 %)	432
CPU views	🔶 🔥 java.security.MessageDigest.getInstance	17,096 bytes (1 %)	525
	👳 🤼 java.io.BufferedReader. <init></init>	16,448 bytes (1 %)	2
	🛛 💁 📤 java.lang.Integer.toHexString	14,400 bytes (1 %)	360
	🗇 🔔 java.util.Set.toArray	12,704 bytes (1 %)	551
	🖕 💁 java.util.LinkedHashSet. <init>(java.util.Collection)</init>	10,800 bytes (1 %)	270
Thread views	or 🏝 java.lang.Object. <init></init>	10,448 bytes (1 %)	448
	🖗 📤 java.lang.StringBuilder.append(java.lang.Object)	10,360 bytes (1 %)	211
	🗇 🔔 java.lang.Integer. <init></init>	9,344 bytes (0 %)	584
200	🗢 🔼 java.lang.StringBuilder.toString	9,104 bytes (0 %)	114
Canadana	🛛 🗢 🦺 java.io.InputStreamReader. <init></init>	8,376 bytes (0 %)	5
M telemetry views	🛛 🗢 🦺 java.lang.Byte. <init></init>	8,192 bytes (0 %)	512
	🗇 🔔 java.util.Set.addAll	8,112 bytes (0 %)	256
	🛛 🗢 🔔 org.apache.commons.logging.Log.error	7,336 bytes (0 %)	5
	🎯 🔔 java.lang.StringBuilder. <init></init>	6,592 bytes (0 %)	98
	👳 🔔 edu.nwu.util.bll.UniqueId.generate	5,456 bytes (0 %)	103
	or 🦺 java.util.LinkedList. <init></init>	4,896 bytes (0 %)	204
Ci La	<ul> <li>A. java.lang.StringBuffer.append(java.lang.String)</li> <li>Java.lang.StringBuffer.toString</li> </ul>	4,784 bytes (0 %)	35 50
	o⊷ 🥼 java.lang.StringBuffer.toString o⊷ 🔥 java.util.Map.put	4,736 bytes (0 %)	156
	or inva.ucii.map.puc	4,488 bytes (0 %) 4,456 bytes (0 %)	97
	or as java.iaig.class.getrieus ⊙r 1. java.util.Set.add	4,416 bytes (0 %)	138
	- 1 java.util.Set.iterator	3,840 bytes (0 %)	120
	O- 1 java.util.List.iterator	3,520 bytes (0 %)	120
9	java.lang.StringBuilder.append(java.lang.String)	3,392 bytes (0 %)	10
	or 23 javanang, bunngbunder, append(javanang, bunng) or ▲ iava util Date <init>0</init>	3 120 hytes (0 %)	130
	View Filters:		how global filters
	All objects Recorded objects Allocation call tree Allocation hot spots		

ession view Pro	filing <u>G</u> o To <u>W</u> indow <u>H</u> elp		
🕹 🕄 😳 😡	📙 🕭 🚳 1 🍇 🗫 1 🎲 🗔 1 🕘 1 📓 1 🎬 🤔	2	
A Memory views	Recorded allocations of: All classes Liveness mode: Live objects		Pa 70 20
, contra , contra	Aggregation level: Methods Hot spot	Filtered classes:     Allocated memory ∠	show separately Allocations
Heap walker CPU views CPU views Thread views WM telemetry views	● Borg.apache.commons.logging.LogFactory.getLog(java.lang.class)         ● 49.9% - 536 kb = 4, 372 alloc. edu.nwu.netmo.examples.MulticastAgent.         ● 10       0.49.9% - 536 kb = 4, 372 alloc. edu.nwu.netb.MaceviveQueue.         ● 10       0.49.9% - 536 kb = 4, 372 alloc. edu.nwu.netb.MaceviveQueue.         ● 10       0.1% - 1, 344 bytes = 36 alloc. edu.nwu.netb.MaceviveQueue.         ● 10       0.1% - 1, 344 bytes = 36 alloc. edu.nwu.netb.MaceviveQueue.         ● 10       0.1% - 1, 208 bytes - 32 alloc. edu.nwu.netb.ML         ● 10       0.1% - 1, 208 bytes - 32 alloc. edu.nwu.net.dll.Pacettagen_register (line: 149         ● 10       0.1% - 1, 208 bytes - 32 alloc. edu.nwu.net.dll.PacettSolete.         ● 10       0.1% - 1, 208 bytes - 32 alloc. edu.nwu.net.dll.PacettSolete.         ● 10       0.1% - 1, 208 bytes - 32 alloc. edu.nwu.net.dll.PacettSolete.         ● 10       0.1% - 1, 208 bytes - 32 alloc. edu.nwu.net.dll.CakeEntry.child         ● 11% - 1, 128 bytes - 15 alloc. edu.nwu.netmo.bll.MacestAgent.main         ● 10       0.1% - 1, 208 bytes - 8 alloc. edu.nwu.netmo.bll.MaticastAgent.main         ● 10       0.1% - 1, 1000 bytes - 8 alloc. edu.nwu.netmo.bll.MaticastAgent.main         ● 10       0.1% - 1, 000 bytes - 8 alloc. edu.nwu.netmo.bll.MaticastAgent.main         ● 10       0.1% - 1, 000 bytes - 8 alloc. edu.nwu.netmo.bll.MaticastAgent.clint>         ● 10       0.1% - 1, 000 bytes - 9 alloc. edu.nw	553 k8 (5 main i) i3 (ine: 118) init>(line: 118) init>(line: 115) thicsatAgent.main clinit> h 410) line: 149) satAgent.publish (line: 187) Agent.publish (line: 152) cefMultcastPublisher.run in	4,626
$\bigcirc$	O 11% - 608 bytes - 19 alloc, edu, nwu, net, wfl, PacketSocket, <init> (     O 10% - 608 bytes - 19 alloc, edu, nwu, nemo, examples. Multicasty     O 0 11% - 608 bytes - 19 alloc, edu, nwu, multicast, bll, Cache, <clinit>      O, 0 11% - 608 bytes - 19 alloc, edu, nwu, nemb ll MamoService, estim (line: 13%)</clinit></init>	Agent.main	C
	View Filters:	Reset view	filters Show global filters

For the subscriber node the view is defined below:

(nemo (3) - JPr		
ession <u>Y</u> iew <u>P</u> rol	ling <u>G</u> o To <u>W</u> indow <u>H</u> elp	
🔊 😳 🖏 🕹	😸 🕭 🚳 a 🎎 🗫 a 🎲 🗔 a 🔘 a 🧱 a 🎆	
*	Recorded allocations of: All classes Liveness mode: Live objects	
Memory views	Aggregation (Cxel: Methods	
Heap walker CPU views Thread views WM telemetry views	□       55:5% + 536 kB - 4,372 alloc. org.apache.commons.logingl.u0g#actory.getLog (line: 59)         □       0:311 + 223 kB - 298 alloc. edu.nwu.net.wfl.PacketSocket. <nit> (line: 115)         □       0:11 + 7% - 113 kB - 1,059 alloc. edu.nwu.net.wfl.PacketSocket. <nit> (line: 115)         □       0:11 + 7% - 113 kB - 1,059 alloc. edu.nwu.net.wfl.PacketSocket. <nit> (line: 115)         □       0:18 - 1,05% - 67,040 bytes - 32 alloc. edu.nwu.net.dll.DatagramSocket. setUp (line: 141)         □       0:19 - 558 bytes - 16 alloc. edu.nwu.net.dll.PacketFactory.<nit> (line: 127)         □       0:19 - 558 bytes - 16 alloc. edu.nwu.net.dll.PacketFactory.<nit> (line: 127)         □       0:19 - 558 bytes - 11 alloc. ;edu.nwu.net.dll.PacketFactory.<nit> (line: 120)         □       0:09 - 266 bytes - 11 alloc.; edu.nwu.net.dll.CustomPacketMapping.<nit> <nit> (line: 120)         □       0:09 - 136 bytes - 3 alloc.; edu.nwu.net.dll.CustomPacketMapping.<nit> <nit> (line: 100)         □       0:09 - 48 bytes - 12 alloc.; edu.nwu.net.bll.BerverSocket.         □       0:09 - 48 bytes - 12 alloc.; edu.nwu.net.bll.ReceiveQuee.         □       0:09 - 48 bytes - 12 alloc.         □       0:18 - 504 bytes - 12 alloc.         □       0:18 - 504 bytes - 12 alloc.</nit></nit></nit></nit></nit></nit></nit></nit></nit></nit>	
(FL)		global filters

) 🔤 💱 🍣		🗟 🗔 + 🞯 + 🎆 🛸					
-	Aggregation level: Classes						
<b>100</b>	Name	Instance count 🙏	Size				
Memory views	char[ ]	5,812	765 kE				
	java.lang.String	3,602	86,448 byte:				
	int[ ]	2,471	161 ki				
	<class>[ ]</class>	2,159	101,696 byte:				
Heap walker	short[ ]	1,360	81,720 byte:				
101	byte[ ]	1,239	344 ki				
-	java.lang.Class	1,029	365 ki				
650	jav Nang, Integer	929	14,864 byte				
	javá util. LinkedHashMap\$Entry	916	29,312 byte				
CPU views	java.util.LinkedHashMap\$KeyIterator	819	26,208 byte				
100	java.util.TreeMap\$Entry	775	24,800 byte				
1	java.util.LinkedList\$Entry	530	12,720 byte				
<u></u>	java.lang.Byte	528	8,448 byte				
Thread views	java.util.HashMap\$Entry	494	11,856 byte				
	java.util.LinkedHashMap	426	20,448 byte				
	java.util.LinkedHashSet	412	6,592 byte				
	java.util.Date	406	9,744 byte				
M telemetry views	java.util.Hashtable\$Entry	377	9,048 byte				
r celemen y views	java.util.Arrays\$ArrayList	308	4,928 byte				
	java.util.HashMap\$KeySet	304	4,864 byte				
	java.lang.StringBuilder	226	3,616 byte				
	edu.nwu.util.bll.Timestamp	214	3,424 byte				
1	java.util.LinkedList	213	5,112 byte				
(215)	java.util.LinkedList\$ListItr	209	6,688 byte				
9	java.util.LinkedHashMap\$EntryIterator	204	6,528 byte				
	java.lang.reflect.Method	151	12,080 byte				
1	java.util.AbstractList\$Itr	102	2,448 byte				
	com.jprofiler.agent.L\$_B	96	1,536 byte				
()	java.util.HashMap	89	3,560 byte				
	java.net.URL	. 89	4,984 byte				
(a)	View Filters:	• *	Reset view filter				

### 4.2 CPU View through Nemo

Method timing is available as a method invocation tree showing percentage of time consumed and absolute time consumed. Following snapshot shows a view of such a tree.

In following given CPU view, again an initial large memory allocation is for the logging feature inbuilt in Nemo This feature is commonly found on all the three node types. The next large memory allocations are for the receive queue and the packet socket. When compared at the method aggregation level, the Nemo BootstrapService.setup is the class with maximum allocation to its methods.

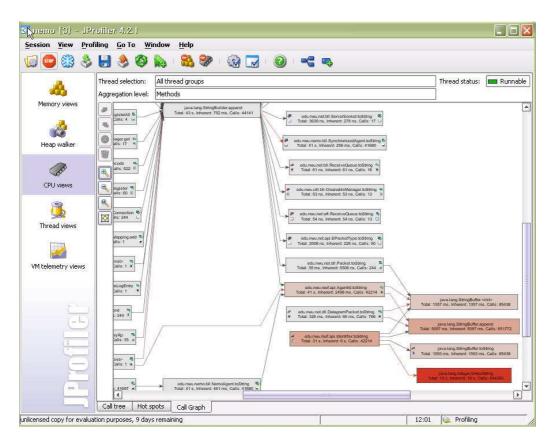
In Publisher node, StreamMulticastAgent.publish is the class with the next dominant allocation. Finally, hot spot in Subscriber type node is allocated for the sendPkt. The next consumption is for the LogFactory.getLog class.

#### 🔀 nemo [3] - JProiiler 4.2.1 Session Yiew Profiling Go To Window Help 😡 🖏 👶 📙 🕭 🥸 🖍 i 🙈 🗫 i 🕅 🗔 i 🥹 Thread selection: All thread groups 💌 Thread status: 📖 Runnable 💌 A Aggregation level: Methods Methods Methods 65.6% - 4,266 ms - 1 inv. edu.nwu.nemo.kamples.MultcastAgent.main 33.0% - 2,147 ms - 1 inv. edu.nwu.nemo.kamples.MultcastAgent.main 93.0% - 2,147 ms - 1 inv. edu.nwu.nemo.kll.PacketHaping.registerAll (line: 99) 9.18.% - 766 ms - 1 inv. edu.nwu.net.dll.PacketHaping.registerAll (line: 99) 9.19.0% - 6,789 µs - 1 inv. edu.nwu.net.dll.PacketHaping.registerAll (line: 97) 9.15% - 6,789 µs - 1 inv. edu.nwu.net.dll.PacketHaping.registerAll (line: 96) 9.12% - 76,315 µs - 1 inv. edu.nwu.net.dll.PacketHaping.registerAll (line: 96) 0.0% - 5,79 µs - 1 inv. edu.nwu.net.dll.PacketHaping.registerAll (line: 96) 9.0% - 5,290 µs - 1 inv. edu.nwu.net.dll.PacketActopicterGatory.create (line: 96) 0.0% - 5,290 µs - 1 inv. edu.nwu.net.dll.PacketActopicterGatory.create (line: 96) 9.0% - 5,290 µs - 1 inv. edu.nwu.net.Mll.SharedPacketSocketActopitsterAll (line: 96) 0.0% - 5,200 µs - 1 inv. edu.nwu.net.Mll.SharedPacketSocketActopitsterGatetActopitster Memory views 8 Heap walker B CPU views na (line: 98) 3 Thread views : 97) U0% - 51 p - 1 m. edu.mwu.nemo.bl.Configuration.getUse.commun.y (mer. 15) 2.7% - 15,4% - 1 m. edu.mwu.net.wlf.PacketSotet..cimt> (ine: 115) 2.7% - 17% - 1 m. yac.in.threAddress.getCalabits (ine: 111) 0.2% - 553 js - 1 m. edu.mwu.net.wlf.PacketSotet..cimt> (ine: 136) 0.7% - 474 us - 1 inv. edu.mwu.net.wlf.PacketSotet..cimt> (ine: 136) 0.7% - 474 us - 1 inv. edu.mwu.net.wlf.PacketSotet..cimt> (ine: 136) 0.7% - 474 us - 1 inv. edu.mwu.net.wlf.PacketSotet..cimt> (ine: 136) 0.7% - 474 us - 1 inv. edu.mwu.net.wlf.PacketSotet..cimt> (ine: 130) 0.7% - 474 us - 1 inv. edu.mwu.net.wlf.PacketSotet..cimt> (ine: 130) 0.7% - 474 us - 1 inv. edu.mwu.net.wlf.PacketSotet..cimt> (ine: 130) 0.7% - 474 us - 1 inv.edu.mwu.net.wlf.PacketSotet..cimt> (ine: 31) 0.7% - 474 us - 1 inv.edu.mwu.net.wlf.PacketSotet..cimt> (ine: 31) 0.7% - 474 us - 1 inv.edu.mwu.net.bl.NenoConfiguration..cimt> (ine: 40) 0.7% - 539 js - 1 m. edu.mwu.net.bl.NenoConfiguration..cimt> (ine: 41) 0.7% - 474 us - 1 inv.edu.mwu.net.bl.NenoConfiguration..cimt> (ine: 41) 0.7% - 474 us - 1 inv.edu.mwu.net.bl.NenoConfiguration..cimt> (ine: 41) 0.7% - 474 us - 1 inv.edu.mwu.net.bl.NenoConfiguration..cimt> (ine: 41) 0.7% - 474 us - 1 inv.edu.mwu.net.bl.NenoConfiguration..cimt> (ine: 133) 0.7% - 474 us - 1 inv.edu.mwu.net.bl.NenoConfiguration..cimt> (ine: 133) 0.7% - 474 us - 1 inv.edu.mwu.net.bl.NenoConfiguration..cimt> (ine: 133) 0.7% - 474 us - 1 inv.edu.mwu.net.bl.NenoConfiguration..cimt> (ine: 133) 0.7% - 474 us - 1 inv.jea.lang.StimpBuffed.met.empt (ine: 157) 0.7% - 774 us - 1 inv.jea.lang.StimpBuffed.met.met.pit(Nen 158) 1 VM telemetry views Reset view filters Show global filters ew Filters: Call tree Hot spots Call Graph unlicensed copy for evaluation purposes, 9 days remaining & Auto-update 5 s 06:57 🐼 Profiling

#### (a) Bootstrap Node

#### (b) Publisher Node

The following snapshot shows a statically calculated thread resolved call graph from main function which is selected in graph nodes. The graph nodes can be methods, classes, packages, or J2EE components, depending on the selected aggregation level. If a graph has been calculated, the context menu also provides access to this action. The resulting graph is static and can be re-calculated be executing Generate graph again[5]. The call graph wizard remembers the last selection. Using this graph, the caller-callee relationship is visibly analyzed. The node color is marked from a gray to red scale for increasing the inherent time and the total time. Therefore, it becomes possible to identify the potential bottlenecks.

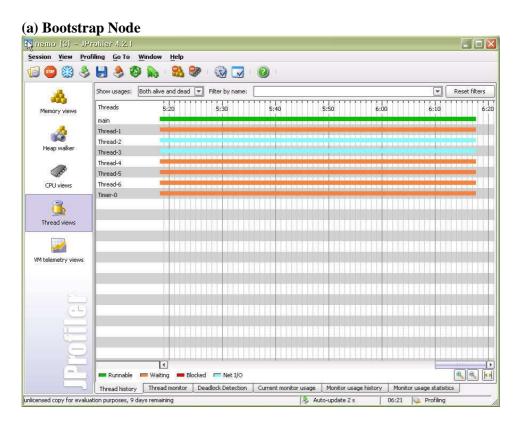


#### (c) Subscriber Node 💽 nemo [3] - JProfiler 4.2.1 Session Yiew Profiling Go To Window Help 🕼 😅 🖏 😓 🛃 🕭 🚳 🖍 i 🎎 🗫 i 🎲 🗔 i 🕑 i 📲 🧠 Thread selection: All thread groups Thread status: 📶 All states 4 Aggregation level: Methods Memory views 10 -0 Heap walker 60 A A A CPU views 3 Thread views 2 VM telemetry views R 4 Call tree Hot spots Call Graph 04:27 🔯 Profiling unlicensed copy for evaluation purposes, 9 days remaining

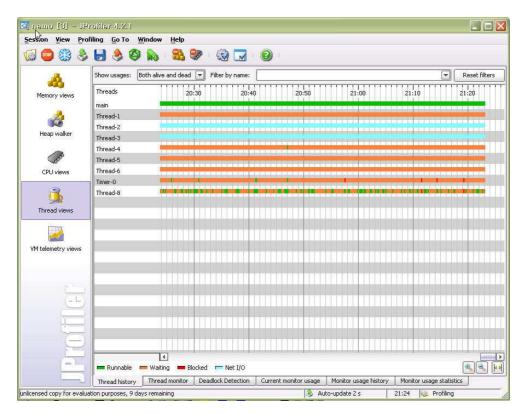
### 4.3 Thread View through Nemo

The threads view set comprises five subviews focusing on past and current thread states (the color scheme of these is as follows: runnable-green, waiting-orange, net I/O-light blue, and blocked-red), past and current monitor usage, and monitor statistics [5]. Following figures show a typical threads view.

The pattern of three nodes are similar to each other with an exception in the case of the thread of publish. There is a provision for an additional thread for the publishing function. In case of short interval, the thread has more control time which is represented by the green color. Based on this view, the thread bottleneck detection is easily identified. As can bee seen the thread for publishing node is in "waiting" state for a longer duration when the interval gets longer. Performance tuning becomes simpler using this view.



### (b) Publisher Node



#### (c) Subscriber Node

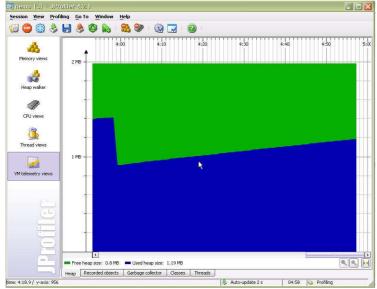
a 😐 🖏 🕹	6 😣 🚳	💫 I 🎎 🌮 I 🤅	🔅 🔲 I 📀 I			
<b>1</b>	Show monitors: [	Waiting and blocking 💌 TI	hreshold in ms: 🛛 💭 Filt	er by name:	Reset fill	ers
	Time 🔺	Duration Type	Monitor ID Monitor clas	ss Waiting thread	Owning thread	_
Memory views	0:01.546	5304 µs 📰 Blocked	4 edu.nwu.net.bll.D	atagr Thread-3 [main]	main [main]	
	0:01.551	980 µs 💳 Blocked	4 edu.nwu.net.bll.D	atagr Thread-4 [main]	main [main]	
-	0:01.559	725 ms 🥅 Waiting	5 edu.nwu.net.bll.D	atagr Thread-4 [main]		
000	0:01.564	821 ms 🥅 Waiting	6 java.util.Collection	ns\$Sy Thread-5 [main]		
Heap walker	0:01.603	327 ms 🥅 Waiting	8 java.util.TaskQue	ue Timer-0 [main]		
10.5	0:01.930	100 ms 🥅 Waiting	8 java.util.TaskQue	ue Timer-0 [main]		
-	0:02.031	100 ms 🥅 Waiting	8 java.util.TaskQue	ue Timer-0 [main]		
(A)	0:02.134	2020 µs 💼 Blocked	9 sun.misc.Launche	r\$App Timer-0 [main]	main [main]	
	0:02.142	1268 µs 🚃 Blocked	9 sun.misc.Launche	r\$App main [main]	Timer-0 [main]	
CPU views	0:02.150	19 ms 💻 Blocked	9 sun.misc.Launche	r\$App Timer-0 [main]	main [main]	
	0:02.173	2562 µs 💳 Blocked	9 sun.misc.Launche	r\$App main [main]	Timer-0 [main]	
	0:02.182	1360 µs 🚃 Blocked	9 sun.misc.Launche	r\$App Timer-0 [main]	main [main]	
22	0:02.185	4202 µs 💳 Blocked	9 sun.misc.Launche	r\$App main [main]	Timer-0 [main]	
Thread views	0:02.197	2122 µs 💼 Blocked	9 sun.misc.Launche	r\$App Timer-0 [main]	main [main]	
	0:02.210	3958 µs 💳 Blocked	9 sun.misc.Launche	r\$App main [main]	Timer-0 [main]	
TTTT I	0:02.214	6830 us 💼 Blocked	9 sun.misc.Launche	r\$App Timer-0 [main]		_
100	Filtered stack trac	e for waiting thread:	ci	tered stack trace for owning thre		-
/M telemetry views						
	edu.nwu.net.wfl. edu.nwu.nemo.dl edu.nwu.nemo.dl edu.nwu.nemo.bl edu.nwu.nemo.bl edu.nwu.nemo.bl edu.nwu.nemo.bl	II.Callback.send(int, edu.nwu II.Callback.heartbeatRp(int, e	nwu.net.api.IMessage, int) er Interface.send(int, edu.nwu.r er .nemo.dll.Packet, int) er idu.nwu.reef.api.AgentId, ed er (edu.nwu.reef.api.AgentId, l	du.nwu.nemo.bll.MulticastAgent. du.nwu.nemo.bll.StreamMulticast.	ic. <init.2(edu.nwu.nemo.bll.configure <init.2(edu.nwu.nemo.bll.configure Agent. <init2(edu.nwu.nemo.bll.co tup(edu.nwu.net,api.IPacketSocke Agent.main(java.lang.String[])</init2(edu.nwu.nemo.bll.co </init.2(edu.nwu.nemo.bll.configure </init.2(edu.nwu.nemo.bll.configure 	ition Infig

### 4.4 VMtelemetry View through Nemo

The VM telemetry view set comprises five different real-time scrolling graphs showing used and free heap space, number of objects (helpfully categorized into arrays and non-arrays), number of loaded classes, garbage collector activity, and number of threads [5]. Following snapshots show this view set.

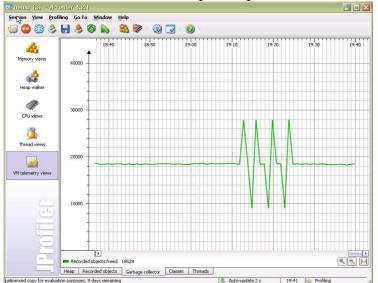
### (a) Bootstrap Node

This graph shows the assigned and freed heap under total 1.98Mbytes. The status of bootstrap node is less dynamic than publisher node.



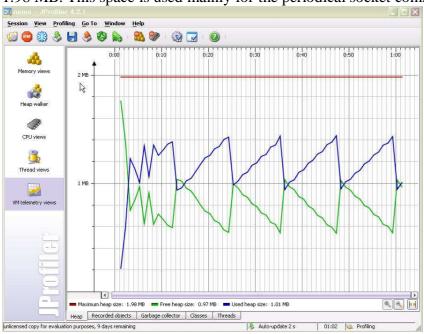
### (b) Publisher Node

This snapshot provides an analysis of the freed objects from the garbage collection function. During the course of object creation and deletion, the shape of graph becomes fluctuating. However, in case the interaction is low there is probably a little of garbage collection. One cannot find a specific pattern between the node types.



### (d) Subscriber Node

The heap status view provides information on the allocated and freed space from a total of 1.98 MB. This space is used mainly for the periodical socket communication.



Lastly, JProfiler's Heap Walker module is the unique aspect based on the easy-to-navigate GUI [5]. The following snapshot view shows the status of classes and arrays in bootstrap nodes. The developer follows the source code easily in run time.

Memory views Mesop walker CPU views CPU views Thread views Wt telemetry telemetry Wt telemetr	asses us	1355 objects in 2: selection step, 239 kB s e selected Name		Instance count A 516 499 446 263 201 202 190 75 75	Size           12, 384 bytes           46, 496 bytes           7, 136 bytes           6, 912 bytes           5, 064 bytes           5, 064 bytes           6, 080 bytes           1, 200 bytes           1, 200 bytes
Theody Water CPU views and CPU	r[] a.lang.Class a.util.Hashtable\$Entry a.util.LinkedList\$Entry a.util.LinkedList\$Entry a.util.LinkedList a.util.LinkedList a.lang.integer a.util.LinkedList a.lang.ref.Finalizer a.util.LinkedList	Name		1516 489 443 263 211 202 190 75 75	12,384 bytes 46,496 bytes 7,136 bytes 10,632 bytes 6,312 bytes 5,064 bytes 16,144 bytes 6,080 bytes 1,200 bytes
Theody Water CPU views and CPU	r[] a.lang.Class a.util.Hashtable\$Entry a.util.LinkedList\$Entry a.util.LinkedList\$Entry a.util.LinkedList a.util.LinkedList a.lang.integer a.util.LinkedList a.lang.ref.Finalizer a.util.LinkedList	ISOIIE		1516 489 443 263 211 202 190 75 75	12,384 bytes 46,496 bytes 7,136 bytes 10,632 bytes 6,312 bytes 5,064 bytes 16,144 bytes 6,080 bytes 1,200 bytes
Theody Water CPU views CPU views Thread views Wt telemetry views Wt telemetry views	r[] a.lang.Class a.util.Hashtable\$Entry a.util.LinkedList\$Entry a.util.LinkedList\$Entry a.util.LinkedList a.util.LinkedList a.lang.integer a.util.LinkedList a.lang.ref.Finalizer a.util.LinkedList			469 446 263 202 190 75 5	46,496 bytes 7,136 bytes 10,632 bytes 6,312 bytes 5,064 bytes 16,144 bytes 6,080 bytes 1,200 bytes
CPU views by CPU v	a.lang.Class a.util.HashMap\$Entry a.util.HashMable\$Entry a.util.LinkedList\$Entry a.util.LinkedHashMap\$Entry a.util.LinkedHashMap a.util.LinkedList a.lang.ref.Finalzer a.util.HashMap			446 263 211 202 190 75	7,136 bytes 10,632 bytes 6,312 bytes 5,064 bytes 16,144 bytes 6,080 bytes 1,200 bytes
CPU views java CPU views java java CPU views java java java java java java java java	a.util.HashMap\$Entry a.util.LinkedList\$Entry a.util.LinkedList\$Entry ass>[] a.util.LinkedHashMap\$Entry a.lang.Integer a.util.LinkedList a.lang.ref.Finalizer a.util.HashMap			443 263 211 202 190 75 75	10,632 bytes 6,312 bytes 5,064 bytes 16,144 bytes 6,080 bytes 1,200 bytes
CPU views java ccPU views java java java Thread views java java yava vM telemetry views java	a.util.Hashtable\$Entry a.util.LinkedList\$Entry ass>[] a.util.LinkedHashMap\$Entry a.lang.Integer a.util.LinkedList a.lang.ref.Finalizer a.util.HashMap			263 211 202 190 75 75	6,312 bytes 5,064 bytes 16,144 bytes 6,080 bytes 1,200 bytes
CPU views jav ccl jav jav jav jav jav jav jav vedu vM telemetry views jav	a.util.LinkedList\$Entry ass>[] a.util.LinkedHashMap\$Entry a.lang.Integer a.util.LinkedList a.util.LinkedList a.util.HashMap			211 202 190 75 75	5,064 bytes 16,144 bytes 6,080 bytes 1,200 bytes
Thread views	ass>[] a.util.LinkedHashMap\$Entry a.lang.Integer a.util.LinkedList a.lang.ref.Finalizer a.util.HashMap			202 190 75 75	16,144 bytes 6,080 bytes 1,200 bytes
Thread views WM telemetry views	a.util.LinkedHashMap\$Entry a.lang.Integer a.util.LinkedList a.lang.ref.Finalizer a.util.HashMap			190 75 75	6,080 bytes 1,200 bytes
Thread views java java java java edu VM telemetry views java	a.util.LinkedList a.lang.ref.Finalizer a.util.HashMap			75	1,200 bytes
VM telemetry views	a.lang.ref.Finalizer a.util.HashMap				
VM telemetry views	a.util.HashMap				
VM telemetry views				5/	2,144 bytes
VM telemetry views java	nun net all CievelaCielal		5	4	2,160 bytes
VM telemetry views java	an waa recaus simplemeta		5	1	1,224 bytes
java	.apache.log4j.CategoryKey		4	5.	736 bytes
	a.security.Provider\$Service#	(ey	4	8	1,056 bytes
java	a.lang.Object		42		336 bytes
java	a.io.File		<b>3</b> 9		624 bytes
sun	.misc.URLClassPath\$FileLoa	der	<b>3</b> 9		624 bytes
java	a.util.regex.Pattern\$Catego	ry	<b>m</b> 36		576 bytes
org	.apache.commons.logging.in	npl.Log4JLogger	29		464 bytes
edu	nwunet.dll.PacketMapping	\$Key	29		464 bytes
	.apache.log4j.Logger		29		1,160 bytes
java	a.security.Provider\$EngineD	escription	24		384 bytes
	a.security.Provider\$UString		23		368 bytes
	a.lang.Package		22		1,056 bytes
	a.security.Provider\$Service		21		1,008 bytes
edu	I.nwu.nemo.core.ENemoPac	ketType	21		504 bytes

# 5. Lessons Learned

Throughout the project, I learned a number of lessons by using JProfiler on an OMP project like Nemo. JProfiler's various output produced enabled to get knowledge on Nemo's setup as an OMP project and also form an opinion on whether Nemo would provide useful support to my team project.

## 5.1 General Characteristics of JProfiler

There are several profiling tools available on the market, such as JProbe and Optimizeit, and to be fair, they all kind of do the same thing. The core features that most end-users are ingested in are the same as other tools - thread monitoring, deadlock detection and memory/class instance monitoring.

JProfiler provides the obvious two advantages as follows.

- Easy to use: The main window is simple and intuitive, allowing a user to quickly navigate between the different views on offer. Filters are also very straightforward and provide a way to focus on a specific set of information.
- Cost: It's one of the 500\$ java products.

Ant it also runs on Mac OS X and provides easy application server integration.

## 5.2 Benefits to the MSE studio project

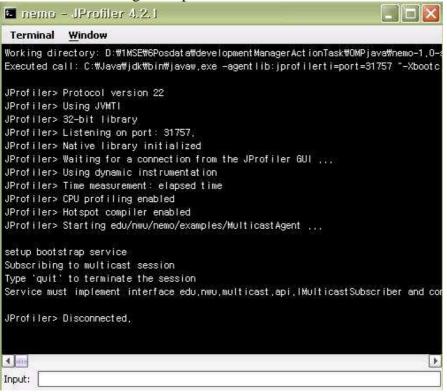
- 1. JProfiler 2.4 is designed to help developers manage performance risks throughout the development process and produce fast, reliable enterprise applications. Developers use profiling technology to identify performance bottlenecks and memory leaks during the development stage of an application. JProfiler is the enterprise-level Java tool available to the development community that integrates CPU, memory and thread profiling in one powerful and robust application.
- 2. Based on the expectations and the results produced by JProfiler, the analysis proved to be beneficial in understanding a simple OMP situation. The same situation was used to make decisions of making Nemo a part of the MSE project.
- 3. The analysis are primarily of 4 views:
  - **Memory View**: JProfiler's memory view section offers dynamically updated views on memory usage and allocations. All views can show live and garbage collected objects.
  - **CPU View**: JProfiler offers various ways to record the call tree to optimize for performance or detail. The thread or thread group as well as the thread status can be chosen for all views.

- **Thread View**: For thread profiling, JProfiler offers thread history, thread monitor, deadlock detection graph, current monitor usage, monitor usage history, and monitor usage statistics.
- VM Telemetry View: To observe the internal state of your JVM, JProfiler offers various telemetry views such as heap, objects, garbage collector, classes and threads.

These profiles provide useful information as described above. For instance, the Logging function is an inherent scheme available in Nemo. Even though the overall performance of Nemo is better than a previous prototype example in terms of NICE (another OMP protocol), it is still a bottleneck in Nemo. Therefore, initial logging function should be designed as an optional function. In conclusion, the one suggestion is that the logging function is replaced by an Aspect-Oriented Program (AOP) which depends upon the analysis result of JProfiler.

## 5.3 Drawbacks

- 1. JProfiler does not provide control over selectively profiling java code in fine-grained level.
- 2. There is no the facility to extract information from the response of a request and use that information in subsequent requests.
- 3. Even though some of multiple executions are done, the disconnected shell script window console still remained like a dangling program as the following snapshot. Therefore, as an improvement step, the related console should disappear when the execution is done and the tool is no longer in operation.



# 6. Conclusions

In this approach, JProfiler as a static analysis tool for OMP project was moderately helpful. The reasons are as follows:

- 1. The tool could provide memory, CPU, and thread view for OMP open product, Nemo. This was when there is a bootstrap, publisher, and subscriber node in the network. This fact is attributed to the real time nature of JProfiler.
- 2. The tool could execute a simple case of an OMP on Nemo and provide insight into the network parameters.
- 3. The various outputs produced with JProfiler enabled to understand the OMP concepts as depicted by Nemo in a better and more productive way. This provides the useful idea on a new design and implementation of OMP in perspectives of time, space and also its related tradeoffs.
- 4. For the short project, the reverse engineering is very important to achieve the basic concept using previous academic or industry product. On that point, this simple and fast usable tool contributes to save the time and money of the project. JProfiler is strongly recommended to the urgent and similar size project.

# 7. References

- 1. http://www.ej-technologies.com/products/jprofiler/tutorials.html
- 2. http://www.javaworld.com/javaworld/jw-08-2003/jw-0822-profiler.html
- 3. <u>http://sourceforge.net/project/showfiles.php?group\_id=160473</u>
- 4. http://weblogs.java.net/blog/simongbrown/archive/2005/02/jprofiler\_minir\_1.html
- 5. JProfiler Manual, ej-technologies, 2006
- 6. CMU MSE Team OMPArchitectability, Mini-Project 1: Tool or Analysis Practicum, Daikon: Invariant Detection of Nemo in association with OMP Project, Spring semester 2006