

Jive

Tool Overview

April 07 :: Spring 2010

Demian Lessa <dlessa@buffalo.edu>

1 Motivation

- 1 Motivation
- 2 Introduction to Jive

- 1 Motivation
- 2 Introduction to Jive
- 3 Jive in Action

- 1 Motivation
- 2 Introduction to Jive
- 3 Jive in Action
- 4 Conclusion

- 1 Motivation
- 2 Introduction to Jive
- 3 Jive in Action
- 4 Conclusion

Traditional Debugging

- Traditional? (e.g., gdb, dbx, WinDbg)
- These debuggers are back-ends, i.e., do all heavy weight-lifting.
- Supports a common debugging strategy: breakpoint, step-inspect loop.

Traditional Debugging

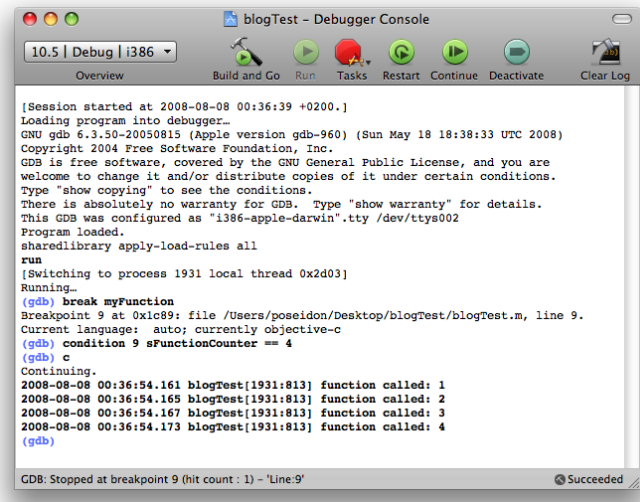
- Traditional? (e.g., gdb, dbx, WinDbg)
- These debuggers are back-ends, i.e., do all heavy weight-lifting.
- Supports a common debugging strategy: breakpoint, step-inspect loop.
- What kind of program data and metadata is available?
 - How is it exposed for inspection?
 - How is it displayed?
 - TUI: limited, no complex information.
 - GUI: delegating to a debugger front-end. (e.g., ddd)

Traditional Debugging

- Traditional? (e.g., gdb, dbx, WinDbg)
- These debuggers are back-ends, i.e., do all heavy weight-lifting.
- Supports a common debugging strategy: breakpoint, step-inspect loop.
- What kind of program data and metadata is available?
 - How is it exposed for inspection?
 - How is it displayed?
 - TUI: limited, no complex information.
 - GUI: delegating to a debugger front-end. (e.g., ddd)
- How is the temporal aspect of program state handled?
 - Only the current state of the program is available!

Traditional Debugging

- Traditional? (e.g., gdb, dbx, WinDbg)
- These debuggers are back-ends, i.e., do all heavy weight-lifting.
- Supports a common debugging strategy: breakpoint, step-inspect loop.
- What kind of program data and metadata is available?
 - How is it exposed for inspection?
 - How is it displayed?
 - TUI: limited, no complex information.
 - GUI: delegating to a debugger front-end. (e.g., ddd)
- How is the temporal aspect of program state handled?
 - Only the current state of the program is available!
- In summary:
 - Benefits: simplicity, familiarity.
 - Limitations: debugging is sequential/procedural in nature; limited visual representation; no support for temporal aspects of the execution.

Figure: gdb Session in the Mac (from <http://blog.timac.org/?p=118>)

```
[Session started at 2008-08-08 00:36:39 +0200.]
Loading program into debugger...
GNU gdb 6.3.50-20050815 (Apple version gdb-960) (Sun May 18 18:38:33 UTC 2008)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-apple-darwin".tty /dev/ttys002
Program loaded.
sharedlibrary apply-load-rules all
run
[Switching to process 1931 local thread 0x2d03]
Running...
(gdb) break myFunction
Breakpoint 9 at 0x1c89: file /Users/poseidon/Desktop/blogTest/blogTest.m, line 9.
Current language:  auto; currently objective-c
(gdb) condition 9 sFunctionCounter == 4
(gdb) c
Continuing.
2008-08-08 00:36:54.161 blogTest[1931:813] function called: 1
2008-08-08 00:36:54.165 blogTest[1931:813] function called: 2
2008-08-08 00:36:54.167 blogTest[1931:813] function called: 3
2008-08-08 00:36:54.173 blogTest[1931:813] function called: 4
(gdb)

GDB: Stopped at breakpoint 9 (hit count : 1) - 'Line:9'
```

- 1 Motivation
- 2 Introduction to Jive**
- 3 Jive in Action
- 4 Conclusion

What is Jive?

- Prototype tool for dynamic program analysis.

What is Jive?

- Prototype tool for dynamic program analysis.
- Jive supports:
 - Traditional debugging.
 - Forward and reverse stepping/skipping.
 - Query-based debugging (guided queries).
 - Visual debugging.

What is Jive?

- Prototype tool for dynamic program analysis.
- Jive supports:
 - Traditional debugging.
 - Forward and reverse stepping/skipping.
 - Query-based debugging (guided queries).
 - Visual debugging.
- Target audience:
 - Software developers (Jive is a development tool).
 - Students and professors (Jive is a pedagogical tool).
 - Researchers (Jive is a research tool).

What is Jive?

- Prototype tool for dynamic program analysis.
- Jive supports:
 - Traditional debugging.
 - Forward and reverse stepping/skipping.
 - Query-based debugging (guided queries).
 - Visual debugging.
- Target audience:
 - Software developers (Jive is a development tool).
 - Students and professors (Jive is a pedagogical tool).
 - Researchers (Jive is a research tool).
- In summary:
 - Benefits: no need to re-execute to return to a previous state; visual model of program execution (enhanced program understanding); declarative queries (higher abstraction of the debugging tasks).
 - Limitations: trace overhead; incremental stepping/skipping back; scalability of diagrams and search queries.

How Jive Works

- Jive gathers data from a Java application running in debug mode.
- Data is received in the form of debug event notifications.
- Jive updates an event data model after every notification.
- Derived models are updated (e.g., object and sequence models).
- Views are updated (e.g., object and sequence diagrams).

Technical Details

- Java based implementation.
- Debugger built on top of JPDA (Java Platform Debugger Architecture).
- Decoupled architecture using the MVC pattern (Model-View-Controller).
- Diagrams built on top of the Eclipse using GEF (Graphical Editing Framework).
- In-memory, Java based data models and query primitives.

Figure: JPDA Overview

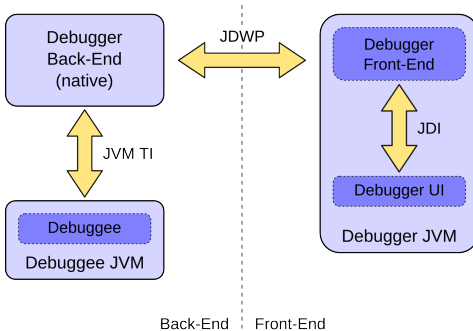


Figure: Jive Architecture Overview

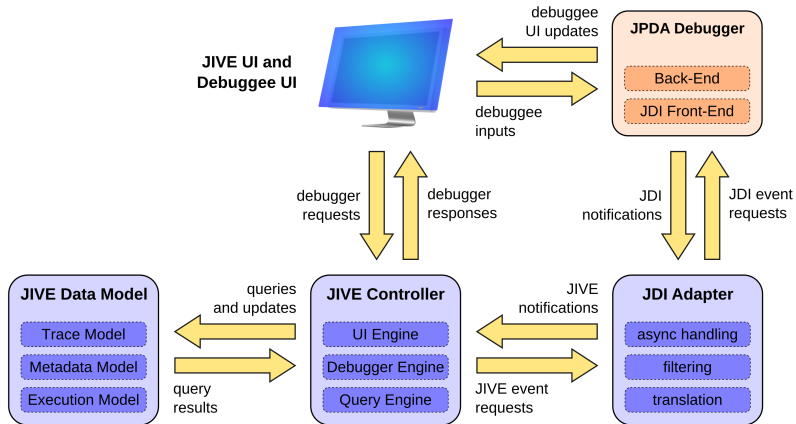
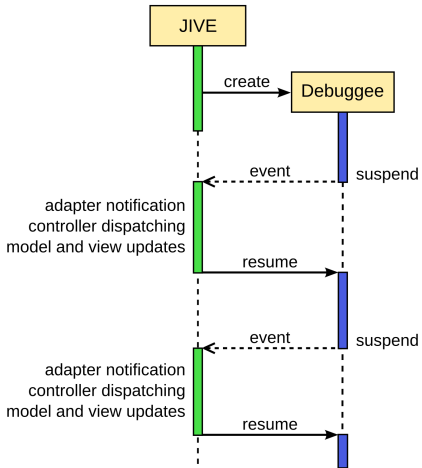


Figure: Jive Interaction with the Debuggee (via JDI)



- 1 Motivation
- 2 Introduction to Jive
- 3 Jive in Action**
- 4 Conclusion

Figure: Dining Philosophers- Initial Setup

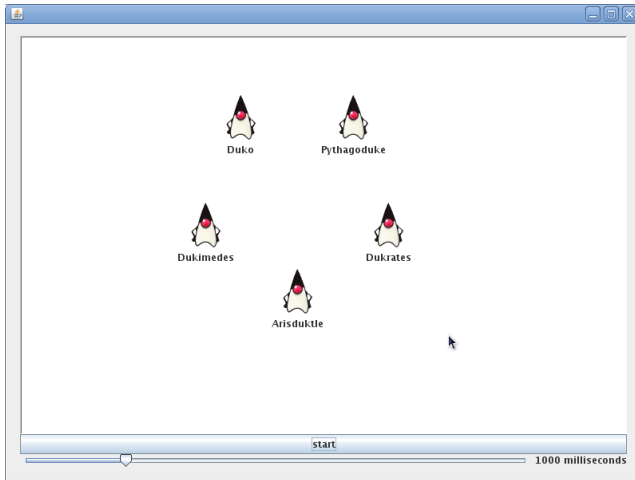


Figure: Dining Philosophers- Philosopher.java

```
1 public class Philosopher implements Runnable {
2
3 public void run() {
4 while (true) {
5     Thread.sleep(Math.random() * grabDelay);
6     clearText();
7     rightStick.grab();
8     setIcon(RIGHTSPOONDUKE);
9
10    Thread.sleep(Math.random() * grabDelay);
11    leftStick.grab();
12    setIcon(BOTHSPoonsDUKE);
13
14    Thread.sleep(Math.random() * parent.grabDelay);
15    rightStick.release();
16    leftStick.release();
17    setIcon(HUNGRYDUKE);
18    setText("Mmmm!");
19
20    Thread.sleep(Math.random() * grabDelay * 4);
21 }
22 }
23 }
```


Figure: Dining Philosophers- Chopstick.java

```
1 public class Chopstick {
2
3     Thread holder = null;
4
5     public synchronized void grab() throws InterruptedException {
6
7         while (holder != null)
8             wait();
9         holder = Thread.currentThread();
10    }
11
12    public synchronized void release() {
13
14        holder = null;
15        notify();
16    }
17
18    public synchronized void releaselfMine() {
19
20        if (holder == Thread.currentThread())
21            holder = null;
22        notify();
23    }
24 }
```

Figure: Dining Philosophers- Object Diagram (Collapsed)

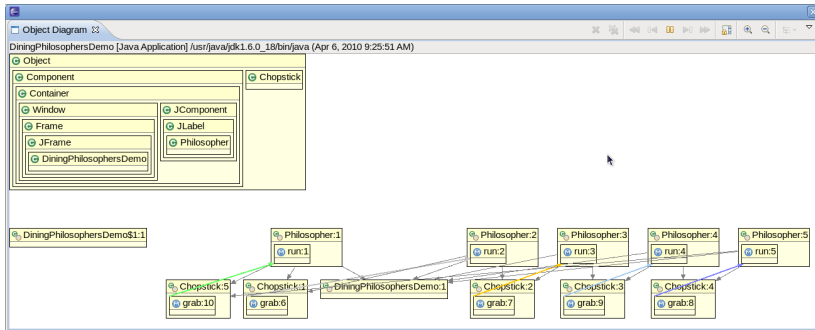


Figure: Dining Philosophers- Object Diagram (Expanded)

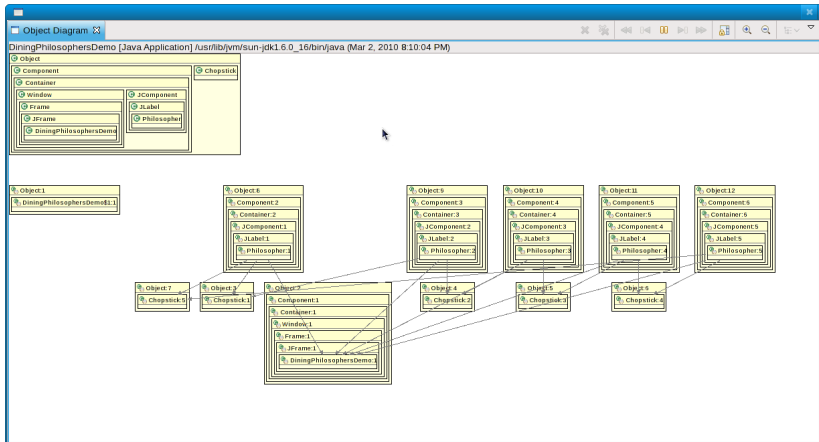


Figure: Dining Philosophers- Interacting

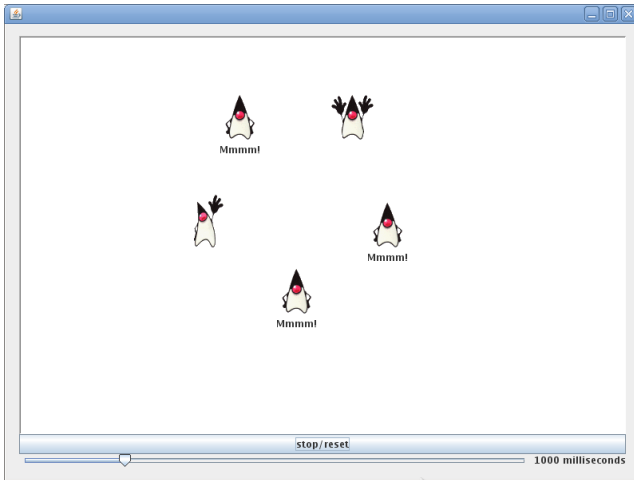


Figure: Dining Philosophers- Sequence Diagram (Interacting)

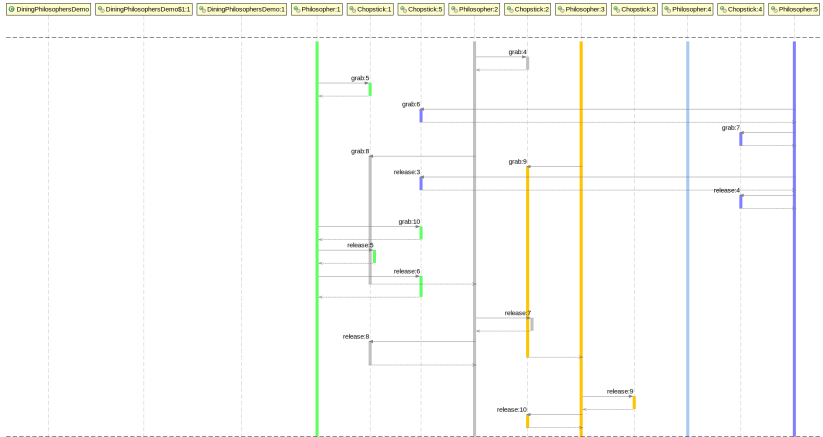


Figure: Dining Philosophers- Deadlocked

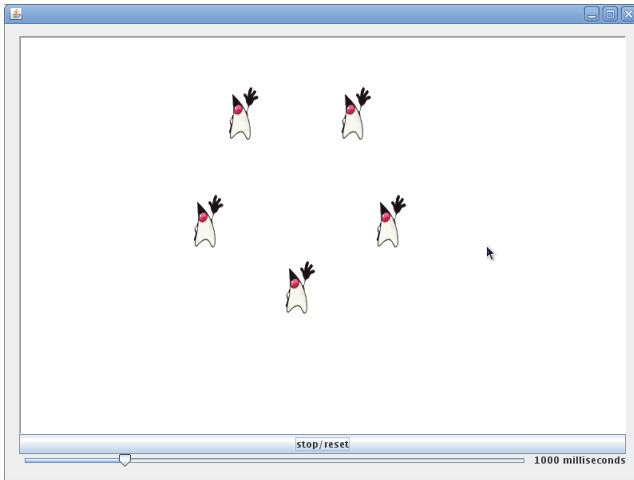


Figure: Dining Philosophers- Sequence Diagram (Deadlocked)

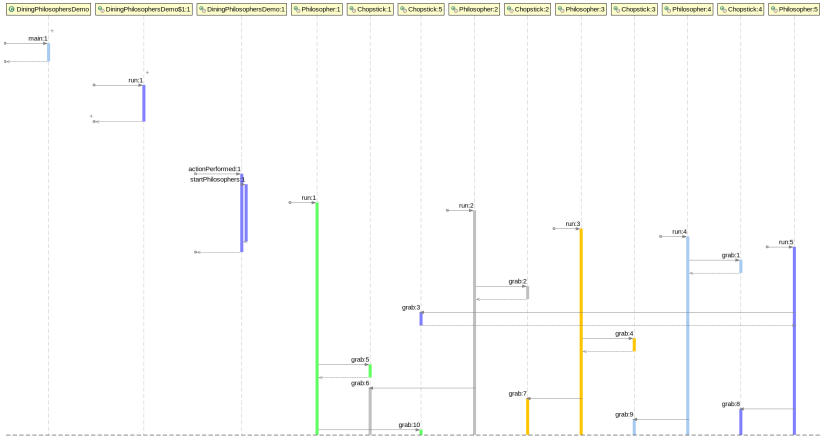


Table: Dining Philosophers- Event Log Snippet

Thread	Event	Type	Details
Thread-2	448	Call Event	target = Chopstick:1#grab:5, actuals = [], caller = Philosopher:1#run:1
Thread-2	449	EOS Event	file = DiningPhilosophersDemo.java, line = 327
Thread-2	450	EOS Event	file = DiningPhilosophersDemo.java, line = 329
Thread-2	451	Assign Event	context = Chopstick:1, variable = holder, value = java.lang.Thread name=Thread-2, id=136)
Thread-2	452	EOS Event	file = DiningPhilosophersDemo.java, line = 330
Thread-2	454	EOS Event	file = DiningPhilosophersDemo.java, line = 293
Thread-2	453	Return Event	returner = Chopstick:1#grab:5, value = <void>
Thread-2	455	EOS Event	file = DiningPhilosophersDemo.java, line = 295
Thread-3	456	EOS Event	file = DiningPhilosophersDemo.java, line = 296
Thread-3	457	Call Event	target = Chopstick:1#grab:6, actuals = [], caller = Philosopher:2#run:2
Thread-3	458	EOS Event	file = DiningPhilosophersDemo.java, line = 327
Thread-3	459	EOS Event	file = DiningPhilosophersDemo.java, line = 328
Thread-4	460	EOS Event	file = DiningPhilosophersDemo.java, line = 296
Thread-4	461	Call Event	target = Chopstick:2#grab:7, actuals = [], caller = Philosopher:3#run:3
Thread-4	462	EOS Event	file = DiningPhilosophersDemo.java, line = 327
Thread-4	463	EOS Event	file = DiningPhilosophersDemo.java, line = 328
Thread-6	464	EOS Event	file = DiningPhilosophersDemo.java, line = 296
Thread-6	465	Call Event	target = Chopstick:4#grab:8, actuals = [], caller = Philosopher:5#run:5
Thread-6	466	EOS Event	file = DiningPhilosophersDemo.java, line = 327
Thread-6	467	EOS Event	file = DiningPhilosophersDemo.java, line = 328

Screencast 1

- Plugin Configuration
- Jive Perspective
- Jive Views
- Debugging with Jive
- Object and Sequence Diagrams
- Sequence Diagram Actions

Screencast 2

- Object Model
- Sequence Model
- Event Log (exporting)
- Guided Search
- Viewing Search Results

- 1 Motivation
- 2 Introduction to Jive
- 3 Jive in Action
- 4 Conclusion**

Status of Jive

- Open source.
- Hosted at Google Code.
- Actively developed.
- Open to new developers.
- Current version supports Eclipse 3.5/Java 1.6.
- Legacy version supports Eclipse 3.4/Java 1.5.