Jive
Tool Overview

April 07 :: Spring 2010

Demian Lessa <dlessa@buffalo.edu>
1 Motivation
1 Motivation

2 Introduction to Jive
Motivation

Introduction to Jive

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)
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).

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.
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

JIVE UI and Debuggee UI

JIVE Data Model
- Trace Model
- Metadata Model
- Execution Model

queries and updates
query results

JIVE Controller
- UI Engine
- Debugger Engine
- Query Engine

JIVE event requests
JIVE notifications

JDI Adapter
- async handling
- filtering
- translation

JPDA Debugger
- Back-End
- JDI Front-End

debuggee UI updates
debugee inputs
debugger requests
debugger responses
JDI notifications
JDI event requests
Figure: Jive Interaction with the Debuggee (via JDI)

- JIVE
- create
- Debuggee
- suspend
- event
- resume
- adapter notification controller dispatching model and view updates
- event
- suspend
- event
- adapter notification controller dispatching model and view updates
- resume
1 Motivation

2 Introduction to Jive

3 Jive in Action

4 Conclusion
Figure: Dining Philosophers- Initial Setup
Figure: Dining Philosophers- Philosopher.java

```java
public class Philosopher implements Runnable {

    public void run() {
        while (true) {
            Thread.sleep(Math.random() * grabDelay);
            clearText();
            rightStick.grab();
            setIcon(RIGHTSPOONDUKE);

            Thread.sleep(Math.random() * grabDelay);
            leftStick.grab();
            setIcon(BOTHSPOONSDUKE);

            Thread.sleep(Math.random() * parent.grabDelay);
            rightStick.release();
            leftStick.release();
            setIcon(HUNGRYDUKE);
            setText("Mmmm!");

            Thread.sleep(Math.random() * grabDelay * 4);
        }
    }
}
```
Figure: Dining Philosophers- Chopstick.java

```java
public class Chopstick {

    Thread holder = null;

    public synchronized void grab() throws InterruptedException {
        while (holder != null)
            wait();
        holder = Thread.currentThread();
    }

    public synchronized void release() {
        holder = null;
        notify();
    }

    public synchronized void releaseIfMine() {
        if (holder == Thread.currentThread())
            holder = null;
        notify();
    }
}
```
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

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