Chronon
A Back-In-Time Debugger For Java

Tobias Sauerwein
Software Analysis – Summer Semester 2011
Overview

- Introduction
- Technical Concept
- Comparison: ObjectFlow Debugger (Paper) – Chronon
- Demonstration
  - General Usage
  - Multi-Threaded Program
- Performance Evaluation
- Conclusion
  - Limitations
  - Advantages
Introduction: What is Chronon?

• Chronon is a commercial Eclipse plugin (Chronon Systems)
• 4 years of development
• Left beta status on April 25, 2011

• Chronon does two things:
  – Recording a program execution
  – Debugging the recording
How do I create a recording?

1. Tell Chronon what classes and packages to record
e.g.:
   – Include: com.mycompany.myproject.** (default)
   – Exclude: java.** (default, can not be removed)

2. Run the program
   • The recording stops once the program is finished.
   • No breakpoints!
How does the recording work? (1/4)

• **What is recorded?**
  “every single line of code” [1] of the included packages/classes
  
  – Variables history
  – Method calls (arguments, return value)
  – Exceptions
  – Console output
  – Threads
How does Chronon collect the data?

- Chronon Recorder is attached to the JVM on startup as **VM agent** (JVM Tool Interface)
  
  ```
  -javaagent:recorder-1.1.0.151.jar=config.txt
  -agentpath:librecorderagent64-1.0.0.so
  ```

- Instruments included classes during load time
- **Logs events to disk**

- To disk? Isn't that kind of slow?
How does the recording work? (3/4)

• **How does Chronon try not to be slow?** [2]
  – Idea: Taking advantage of the hardware
    • MultiCore (increasing number of cores)
    • Memory is cheap
    • Disks are getting faster (Solid State Drives – SDD)
  – Only collect minimal amount of data needed during recording
  – Minimal work inside application threads
  – Recorded data is stored in a **buffer in memory**
  – **Flusher threads** write buffer to disk in highly compressed file (gets unpacked for debugging)
How does the recording work? (4/4)

Application Threads

Memory Buffer

Recorded Data

Flusher Threads

Recording

Application running with Chronon

Image Source: [2]
Comparison: "Practical Object-Oriented Back-In-Time Debugging", Lienhard et al. (ObjectFlow Debugger) [3]

- **Short recap: What approach did the paper take?**
  - Extends the VM: object references are represented as real objects (*alias objects*) on the heap
  - Alias objects keep track of the object flow and field history
  - Only keeps the history of those objects that are referenced in the current program state (using standard VM garbage collection)
## Comparison: ObjectFlow Debugger – Chronon (1/2)

<table>
<thead>
<tr>
<th><strong>ObjectFlow Debugger</strong></th>
<th><strong>Chronon</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Only history of referenced objects</td>
<td>Complete history</td>
</tr>
<tr>
<td>History in memory</td>
<td>History on disk</td>
</tr>
<tr>
<td>“Live” debugging (breakpoints, modify variables/code)</td>
<td>Debugging on a recording (no breakpoints)</td>
</tr>
<tr>
<td>History up to a breakpoint</td>
<td>History of whole program run</td>
</tr>
<tr>
<td>Also tracks reading a field and adding objects to an array</td>
<td>Only tracks assignments</td>
</tr>
<tr>
<td>Deep modifications to the VM</td>
<td>Runs with every JVM</td>
</tr>
</tbody>
</table>
Comparison: ObjectFlow Debugger – Chronon (2/2)

<table>
<thead>
<tr>
<th>ObjectFlow Debugger</th>
<th>Chronon</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Focus on object flow</strong></td>
<td><strong>Focus on control flow (time line)</strong></td>
</tr>
<tr>
<td>• <em>Where was this object used (as field, argument, return value)</em>?</td>
<td>• <em>Which values did this variable have</em>?</td>
</tr>
<tr>
<td>• <em>How was the object passed into this method</em>?</td>
<td>• <em>When was this method called</em>?</td>
</tr>
<tr>
<td></td>
<td>• <em>Which statement was executed after/before this statement</em>?</td>
</tr>
</tbody>
</table>
Demonstration

• Debugging mouse-events
  – Using the conventional debugger
  – Using Chronon

• Debugging a multi-threaded application
Chronon Debugger

- Timeline
- Run to line/method (Forward/Backward in time)
- Calls on the current line (arguments, return value)
- Thrown exceptions
- Variable history
- Method history
- Recorded console
- Stacktrace
- Threads
Performance-Evaluation

• Two tests with open-source Java projects
  – Java SE application: *Object database db4o* [4] (database file on disk)

• Micro-Benchmark
Performance: db4o

• Test setup
  – Unit-Tests: com.db4o.db4ounit.common.btree.AllTests
  – Chronon Include-Filter: com.db4o.**
  – Quad core (4 x 2.5 GHz), 2 flusher threads

• Results
  – Time
    • Without Recording: ~ 44 s
    • With Recording: ~ 130 s (factor: 3)
  – Recording size
    • Packed: 28 MB
    • Unpacked: 583 MB (time to unpack: ~ 30 min!)
  – Time “steps” (events): ~ 9.8 millions
Performance: Google Refine (Java EE)

- Test setup
  - Chronon Include-Filter: com.google.refine.**
  - Test case: Sorting a table
  - Measured **time to receive JSON response** in Google Chrome Developer Tools

- Results
  - Avg. time
    - Without Recording: **0.57 s**
    - With Recording: **1.09 s (factor: 2)**
package chronontests.micro;

public class Test {
    private static int COUNT = 1000000;

    public static void main(String[] args) {
        System.out.println("Start");

        for (int i = 0; i < COUNT; i++) {
            A obj = new A(i, i + 1);
            //obj.test();
            System.out.println(obj.test());
        }

        System.out.println("End");
    }
}

public static class A {
    private int a, b;

    public A(int a, int b) {
        this.a = a;
        this.b = b;
    }

    public int test() {
        return a + b;
    }
}
Performance: Micro-Benchmark (2/2)

• Results (total time)
  – Without printing the result of “obj.test()”
    • Without Recording: 0.067 s
    • With Recording: 15.46 s (factor: 230!)

  – With printing the result of “obj.test()”
    • Without Recording: 15.5 s
    • With Recording: 26.4 s (factor: 1.7)
Using Chronon for Java EE applications

• “57% of application performance spent in data access”

• Typical Java EE applications spent a large amount of their execution time waiting on resources (databases, IO, network).

  → Chronon does not record new data during that time
Limitations (1/2)

• Inspecting collections and other classes of external libraries (planned: “lightweight instrumentation” of classes, only record changes to fields)

• Always records the whole program execution

• Unpacking a recording takes its time
Limitations (2/2)

• No automated **dynamic slicing** [6]
  - Slice: a **subset of a program relevant for a given statement** $S$
    • Forward slices: all statements influenced by $S$
    • Backward slices: all statements that influence $S$
  - Data/Control dependencies
    • **Where does this value go to?**
    • **Where does this value come from?**
    • **Why is this line of code executed?**
    • **Why was this line of code not executed?**

```c
int main() {
    int a, b, sum, mul;
    sum = 0;
    mul = 1;
    a = read();
    b = read();
    while (a <= b) {
        sum = sum + a;
        mul = mul * a;
        a = a + 1;
    }
    write(sum);
    write(mul);
}
```

Backward slice for `write(mul)`
Advantages (1/4)

• Helps narrowing down the defect which caused a failure
  – “Debugging is a search in space and time” [6]
    → Going backwards in time is extremely useful
  – Eliminates problems of breakpoint debuggers [7]
    • No “guessing” where to put the breakpoint
    • No “Whoops, I went too far.”

Image Source: [6]
Advantages (2/4)

- Gives direct answers to questions like ...
  - *When was this value set?*
  - *Who set this value?*
  - *When did this output happen?*
  - *When was this exception thrown?*
  - *Has this line of code been executed?*
  - ...
Advantages (3/4)

- Makes **reproducing problems** easier
  - Record the program in an environment where the problem occurs
  - Analyze the recording on a development system

- Reproducing data / user interaction / communication / operating environments / schedules

![Diagram of input sources and replay](Image Source: [6])
Advantages (4/4)

• Helps debugging multi-threaded applications

• Useful tool for code comprehension

• Scales with the hardware

• Highly compressed recording that can be exchanged between developers

• Reasonable overhead while recording
Conclusion

• Chronon is an useful tool that makes debugging easier

• Analyzing the recording is already powerful, but still room for improvements
How can I give it a try?

- Chronon Systems:
  http://www.chrononsystems.com/

- Licenses
  - 30 day evaluation license
  - 1 year student license
Related Work / Software

- IntelliTrace in Visual Studio 2010 Ultimate: C#, VB, ASP.NET, F#

- Omniscient Debugging (ODB) for Java, Bill Lewis (last update: February 2007), similar to Chronon: http://lambdacs.com/debugger/debugger.html


References / Links

[1] Chronon:  
http://www.chrononsystems.com/

http://eblog.chrononsystems.com/design-and-architecture-of-the-chronon-record-0#!/


[4] Object database db4o:  
http://www.db4o.com/

[5] Google Refine:  
http://code.google.com/p/google-refine/


Discussion

Thanks for your attention!

Any questions?