EASEly extending Eclipse Trace Compass

Bernd Hufmann, Ericsson AB
Geneviève Bastien, Polytechnique Montréal
Agenda

— Background
— Trace Compass and its extensibility
— Trace Compass and EASE
— Demo
— Q&A
What is tracing?

- Trace
  - Series of events over time
  - Event collected at tracepoints during program execution
  - Each event has a type and payload
- Use the events as input for analysis
- Create visualization graphs with these analysis
- Tracing use cases
  - Profile application
  - Find long executions
  - Investigate real-time deadlines
  - Find memory or load issues
  - Investigate concurrency problems
**Eclipse Trace Compass** is an open source application to solve performance and reliability issues by reading and analyzing traces and logs of a system.

Its goal is to provide views, graphs, metrics, and more to help extract useful information from traces, in a way that is more user-friendly and informative than huge text dumps.

Key characteristics
- Handles trace larger than available memory
- Correlates traces from heterogenous system
Why is extensibility important for us?
Why is extensibility important for us?

“One of these things is not like the others, one of these things is not the same” (Sesame Street)

Application One ≠ Application Two
Trace format A ≠ Trace format B
Analysis Foo ≠ Analysis Bar
Graph Baz ≠ Graph Qux
Problem X ≠ Problem Y
Why is extensibility important for us?

“One of these things is not like the others, one of these things is not the same” (Sesame Street)

<table>
<thead>
<tr>
<th>Application</th>
<th>One</th>
<th>≠</th>
<th>Application</th>
<th>Two</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace format</td>
<td>A</td>
<td>≠</td>
<td>Trace format</td>
<td>B</td>
</tr>
<tr>
<td>Analysis</td>
<td>Foo</td>
<td>≠</td>
<td>Analysis</td>
<td>Bar</td>
</tr>
<tr>
<td>Graph</td>
<td>Baz</td>
<td>≠</td>
<td>Graph</td>
<td>Qux</td>
</tr>
<tr>
<td>Problem</td>
<td>X</td>
<td>≠</td>
<td>Problem</td>
<td>Y</td>
</tr>
</tbody>
</table>

But many re-usable concepts!
Why is extensibility important for us?

“One of these things is not like the others, one of these things is not the same”

Application One ≠ Application Two
Trace format A ≠ Trace format B
Analysis Foo ≠ Analysis Bar
Graph Baz ≠ Graph Qux
Problem X ≠ Problem Y

But many re-usable concepts!

Give flexibility to the users
Trace analysis and visualization flow

- **Trace(s)**
  - Parse
- **Stream of events**
  - Analyze
- **Persistence**
  - Query
- **Views**
  - Events Table
  - Intermediate Results
Trace analysis and visualization flow

1. **Trace(s)**
2. **Stream of events**
3. **Persistence**
4. **Views**

- **Extensions**
  - parse
  - analyze
  - query

**Events Table**

**Intermediate Results**

**Views**
Trace parser extensibility

- Extension point to define trace parsers
  - Automatic trace type detection
  - UI to select and manage trace types

- Java APIs for parsers

- Build-in parsers, e.g. Common Trace Format (CTF)

- Custom Text and XML parser wizard
  - Create parser on-the-fly from UI
Analysis extensibility

- Extension point to define trace analysis
  - Manage analysis in Eclipse
  - Show available analysis in UI

- Java APIs

- Data persistence
  - state system
  - segment store

- XML defined analysis
  - Create them on-the-fly

XML definitions
Visualization extensibility

- Java APIs for common views
- Default implementations
- Re-usable widgets
- XML defined views
  - Create them on-the-fly

XML definitions
In-depth: XML analysis and views

- Find a sequence of data within a trace
- Generate state systems
- Do timing analysis
- Define specialized views
- Pattern analysis
**XML analysis and views**

**Challenges**

- Finite number of available analyses
- Some flexibility with XML analyses
- Very verbose
- Hard to read
- Hard to debug
- But it works!
XML analysis and views

Challenges

- Finite number of available analyses
- Some flexibility with XML analyses
- Very verbose
- Hard to read
- Hard to debug
- But it works!
- Ultimate flexibility: Scripting
EASE — Eclipse Advanced Scripting Environment

EASE Core: Integrates scripting in Eclipse

Script engines
- Nashorn (js)
- Python
- JRuby
- Rhino (js)
- Groovy

Modules
- Provided by Trace Compass
  - Analysis
  - Util
  - Data Provider
  - Filters
  - View
  - Trace/Trace UI
- Others
EASE highlights

- Execute scripts in context of Eclipse
- Supports multiple scripting languages
- Provide access to your application classes (java)
- Interact with your workbench
- Integrate scripts into UI (e.g. toolbar, menu) using keywords
- Extend easily existing application
- Rapid prototyping
- Plug-in mechanism to add custom modules
- Easily share scripts
Example scripts: Demo

```c
int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);
    // Find out rank, size
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    int token;
    // Receive from the lower process and send to the higher process. Take care
    // of the special case when you are the first process to prevent deadlock.
    if (world_rank == 0) {
        MPI_Recv(&token, 1, MPI_INT, world_rank - 1, 0, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
        printf("Process %d received token \%d from process \%d
", world_rank, token,
                world_rank - 1);
    } else {
        // Set the token's value if you are process 0
        token = -1;
    }
    MPI_Send(&token, 1, MPI_INT, (world_rank + 1) % world_size, 0,
             MPI_COMM_WORLD);
    // Now process 0 can receive from the last process. This makes sure that at
    // least one MPI_Send is initialized before all MPI_Recevs (again, to prevent
    // deadlock)
    if (world_rank == 0) {
        MPI_Recv(&token, 1, MPI_INT, world_size - 1, 0, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
        printf("Process %d received token \%d from process \%d\n", world_rank, token,
                world_size - 1);
    }
    MPI_Finalize();
}
```

Instrumented application with custom tracepoints
Example scripts: Demo

```c
int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);
    // Find out rank, size
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    int token;
    // Receive from the lower process and send to the higher
    // of the special case when you are the first process
    if (world_rank == 0) {
        MPI_Recv(token, 1, MPI_INT, world_rank - 1, 0, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
        printf("Process %d received token %d from process \n", world_rank - 1);
    } else {
        // Set the token's value if you are process 0
        token = -1;
    }
    MPI_Send(token, 1, MPI_INT, (world_rank + 1) % world_size, 0,
             MPI_COMM_WORLD);
    // Now process 0 can receive from the last process. This makes sure that at
    // least one MPI_Send is initialized before all MPI_Recvs (again, to prevent
    // deadlock)
    if (world_rank == 0) {
        MPI_Recv(token, 1, MPI_INT, world_size - 1, 0, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
        printf("Process %d received token %d from process %d\n", world_rank, token,
               world_size - 1);
    }
    MPI_Finalize();
}
```

The view we want to get
Example scripts: Demo

```javascript
// Load the proper modules
loadModule("/TraceCompass/Trace")

// Get the active trace
var trace = getActiveTrace()

// Get an event iterator on that trace
var iter = getEventIterator(trace)

// Iterate through the events
var event = null
while (iter.hasNext()) {
    event = iter.next()

    // For each event, print the name and the field names
    eventString = event.getName() + " --> { 

    var fieldsIterator = event.getContent().getFieldNames().iterator()
    while (fieldsIterator.hasNext()) {
        eventString += fieldsIterator.next() + " 
    }
    eventString += "}"

    print(eventString);
}

Step 1:
Read events from the trace
```
Example scripts: Demo

```javascript
// Get an event iterator on that trace
var iter = getEventIterator(trace);

// Associate a TID with an MLP worker
var tidWorkerMap = {};

// Iterate through the events
var event = null;
while (iter.hasNext()) {
  event = iter.next();

  // For each event, print the name and the field names
  eventString = event.getName() + "\n"
  var fieldsIterator = event.getContentType().getFieldNames().iterator();
  while (fieldsIterator.hasNext()) {
    eventString += fieldsIterator.next() + "\n";
    if (eventName == "ring:initialize") {
      tid = getEventFieldValue(event, "contextudad")
      worker_id = getEventFieldValue(event, "worker_id")
      tidWorkerMap[tid] = worker_id
      print("TID: tid = "+tid + ", worker_id = "+worker_id)
    } else if (eventName == "ring:recv_entry") {
      tid = getEventFieldValue(event, "contextudad")
      worker_id = tidWorkerMap[tid]
      print("Entering Reception \n TID: tid = "+tid + ", worker_id = "+worker_id)
    } else if (eventName == "ring:recv_exit") {
      tid = getEventFieldValue(event, "contextudad")
      worker_id = tidWorkerMap[tid]
      source = getEventFieldValue(event, "source")
      print("Exiting Reception \n TID: tid = "+tid + ", worker_id = "+worker_id + ", source = "+source)
    } else if (eventName == "ring:send_entry") {
      tid = getEventFieldValue(event, "contextudad")
      worker_id = tidWorkerMap[tid]
      dest = getEventFieldValue(event, "dest")
      print("Entering Send \n TID: tid = "+tid + ", worker_id = "+worker_id + ", dest = "+dest)
    } else if (eventName == "ring:send_exit") {
      tid = getEventFieldValue(event, "contextudad")
      worker_id = tidWorkerMap[tid]
      print("Exiting Send \n TID: tid = "+tid + ", worker_id = "+worker_id)
    }
  }
  eventString += "}\n"
  console.log(eventString);
}
```

Step 2:
For each event, get the event field values
Example scripts: Demo

```
// Load the proper modules
loadModule("/TraceCompass/Trace");
loadModule("/TraceCompass/Analysis");

// Get the active trace
var trace = getActiveTrace();
var id = 144 +33,11; var iter = getEventIterator(trace);
// Associate a TID with an mpl worker
var tidToWorkerMap = {};

// Get an analysis
var analysis = createScriptedAnalysis(trace, "ringTimeline.js");

// Get the analysis’s state system so we can fill it, false indicates to create a new state system even if
// there is one already.
var ss = analysis.getStateSystem(false);

// Iterate through the events
var event = null;
while (iter.hasNext()) {
  event = iter.next();
  if (event.type == "new") {
    tid = getEventFieldValue(event, "context_tid");
    worker_id = getEventFieldValue(event, "worker_id");
    tidToWorkerMap[tid] = worker_id;
    print("Init -> tid: " + tid + ", worker id: " + worker_id);
  } else if (event.type == "recv_entry") {
    tid = getEventFieldValue(event, "context_tid");
    worker_id = tidToWorkerMap[tid];
    print("Entering Reception -> tid: " + tid + ", worker id: " + worker_id);
    // Save the state of the resource as waiting for reception
    quark = ss.getQuarkAbsoluteAndAdd(worker_id);
    ss.modifyAttribute(event.getTimestamp()).toNanos(), "Waiting for reception", quark);
  } else if (event.type == "recv_exit") {
    tid = getEventFieldValue(event, "context_tid");
    worker_id = tidToWorkerMap[tid];
    print("Exiting Reception -> tid: " + tid + ", worker id: " + worker_id);
    // Remove the waiting for reception state
    quark = ss.getQuarkAbsoluteAndAdd(worker_id);
    ss.removeAttribute(event.getTimestamp()).toNanos(), quark);
  }
}

// Done parsing the events, close the state system at the time of the last event, it needs to be done manually
if (event != null) {
  ss.closeHistory(event.getTimestamp().toNanos());
}
```

Step 3:
Save the states in a state system (backend from which views get their data)
Step 4:
Show a time graph view for the states

Example scripts: Demo

```java
// Load the proper modules
loadModule("/TraceCompass/Trace")
loadModule("/TraceCompass/Analysis")
+loadModule("/TraceCompass/DataProvider")
+loadModule("/TraceCompass/View")

// Get the active trace
var trace = getActiveTrace()
@@ -63,3 +65,13 @@ while (iter.hasNext()) {
    if (event != null) {
        ss.closeHistory(event.getTimestamp().toNanos());
    }
+    +//Get a time graph provider from this analysis, displaying all attributes
+    +//Create a map and fill it, because javascript map cannot use the EASE constants as keys
+    +var map = new java.util.HashMap();
+    +map.put(ENTRY_PATH, '*');
+    +provider = createTimeGraphProvider(analysis, map);
+    +if (provider != null) {
+        +// Open a time graph view displaying this provider
+        +openTimeGraphView(provider);
+    }
```
Example scripts: Demo

See the Trace Compass EASE Tutorial lab for complete diff and script

Step 5:
Add the arrows

```javascript
[...]
+ duration = arrow["endTime"] - startTime;
+ // Add the arrow to the arrows list
+ tgArrows.getList().add(createArrow(srcId, dstId, startTime, duration,
+) +
+ // A function used to return the entries to the data provider. It receives the
+function getEntries(parameters) {
+   // The list is static once built, return all entries
+   return tgEntries.getList();
+ } +
+ // A function used to return the arrows to the data provider. It receives the
+function getArrow(parameters) {
+   // Just return all the arrows, the view will take those in the range
+   return tgArrows.getList();
+ } +
+ // Create a scripted data provider for this analysis, using script functions to get the entries, row model
+provider = createScriptedTimeGraphProvider(analysis, getEntries, null, getArrows);
+ if (provider != null) {
+   // Open a time graph view displaying this provider
+   openTimeGraphView(provider);
How-to install

Trace Compass add-ons

- In RCP use “Menu “Tools -> Add-ons…”
- In Eclipse IDE use: Update Site
Takeaways

— Trace Compass is extensible

— EASE provides scripting capabilities to Eclipse applications

— Perfect match: EASE and Trace Compass

— EASE Scripting is now available for Trace Compass users

— Quickly create your own script and share it with colleagues (and community 😊)
References

— Trace Compass
  — http://tracecompass.org
  — https://projects.eclipse.org/projects/tools.tracecompass
  — https://projects.eclipse.org/projects/tools.tracecompass.incubator

— Trace Compass Scripting
  — http://versatic.net/tracecompass/introducingEase.html
  — https://github.com/tahini/tracecompass-ease-scripting
  — https://github.com/tuxology/tracevizlab/

— Eclipse EASE
  — https://projects.eclipse.org/projects/technology.ease
Contacts

— Presenter
  — Geneviève Bastien: gbastien@versatic.net
  — Bernd Hufmann: bernd.hufmann@ericsson.com

— Mailing list
  — tracecompass-dev@eclipse.org

— IRC
  — oftc.net #tracecompass