

Visually representing data-driven analysis using state diagrams

Simon Delisle

Michel Dagenais

Progress Report Meeting
Dec 2014



**POLYTECHNIQUE
MONTRÉAL**

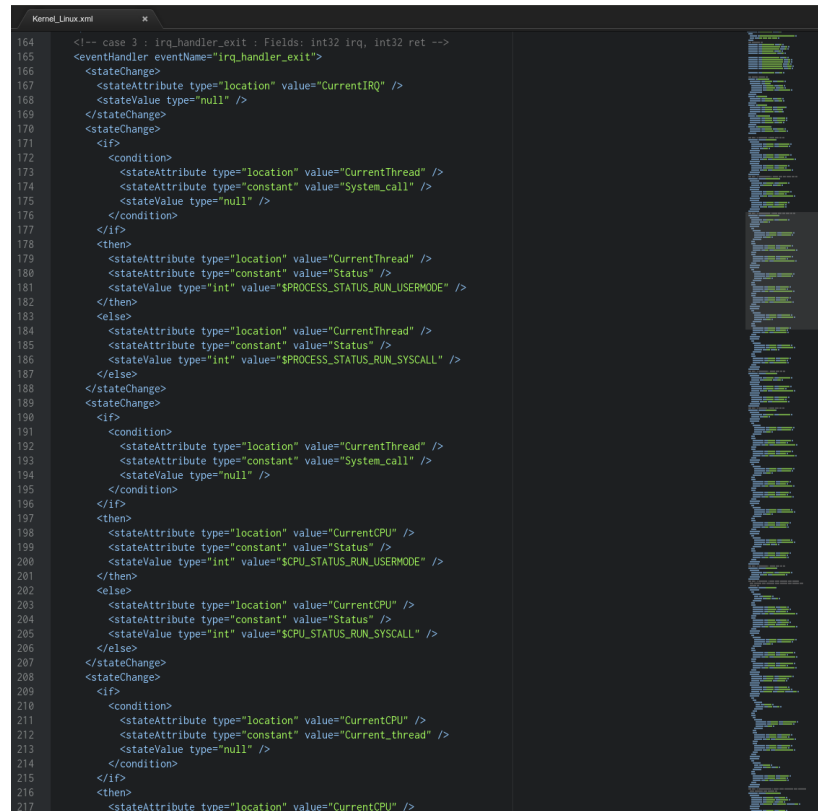
LE GÉNIE
EN PREMIÈRE CLASSE

Presentation plan

- Introduction
- Related work
- Proposed solution
- Results
- Demo
- Road ahead

Introduction

- Java state provider -> XML state provider
- It might be difficult for some users to deal directly with the XML
- We need a simple UI to define those things



```
Kernel_Linux.xml
164 <!-- case 3 : irq_handler_exit : Fields: int32 irq, int32 ret -->
165 <eventHandler eventName="irq_handler_exit">
166   <stateChange>
167     <stateAttribute type="location" value="CurrentIRQ" />
168     <stateValue type="null" />
169   </stateChange>
170   <stateChange>
171     <IF>
172       <condition>
173         <stateAttribute type="location" value="CurrentThread" />
174         <stateAttribute type="constant" value="System_call" />
175         <stateValue type="null" />
176       </condition>
177     </IF>
178   </then>
179   <stateAttribute type="location" value="CurrentThread" />
180   <stateAttribute type="constant" value="Status" />
181   <stateValue type="int" value="$PROCESS_STATUS_RUN_USERMODE" />
182 </then>
183 <else>
184   <stateAttribute type="location" value="CurrentThread" />
185   <stateAttribute type="constant" value="Status" />
186   <stateValue type="int" value="$PROCESS_STATUS_RUN_SYSCALL" />
187 </else>
188 </stateChange>
189 <stateChange>
190   <IF>
191     <condition>
192       <stateAttribute type="location" value="CurrentThread" />
193       <stateAttribute type="constant" value="System_call" />
194       <stateValue type="null" />
195     </condition>
196   </IF>
197   <then>
198     <stateAttribute type="location" value="CurrentCPU" />
199     <stateAttribute type="constant" value="Status" />
200     <stateValue type="int" value="$CPU_STATUS_RUN_USERMODE" />
201   </then>
202 <else>
203   <stateAttribute type="location" value="CurrentCPU" />
204   <stateAttribute type="constant" value="Status" />
205   <stateValue type="int" value="$CPU_STATUS_RUN_SYSCALL" />
206 </else>
207 </stateChange>
208 <stateChange>
209   <IF>
210     <condition>
211       <stateAttribute type="location" value="CurrentCPU" />
212       <stateAttribute type="constant" value="Current_thread" />
213       <stateValue type="null" />
214     </condition>
215   </IF>
216   <then>
217     <stateAttribute type="location" value="CurrentCPU" />
```

Introduction - Goal

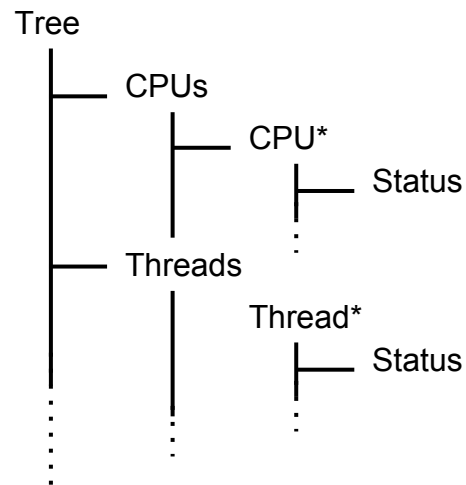
- Capturing in a convivial way all the information related to data-driven trace analysis

Related work

- State provider
- Attribute tree
- Graphiti and Eclipse Modeling Framework (EMF)

Attribute tree

- Each attribute contains a state value
 - Each attribute node represents a system resource
- resource



Graphiti and EMF

- Graphiti is a graphic framework
- Editor for domain models like EMF
- We could have also used another graphic framework like Sirius

Solution

- Generate the actual XML with a modeling tool
- Develop a state machine model
 - Adapted to tracing
 - Based on UML
- Use Graphiti to manipulate this model
- Define an attribute tree for easier modeling

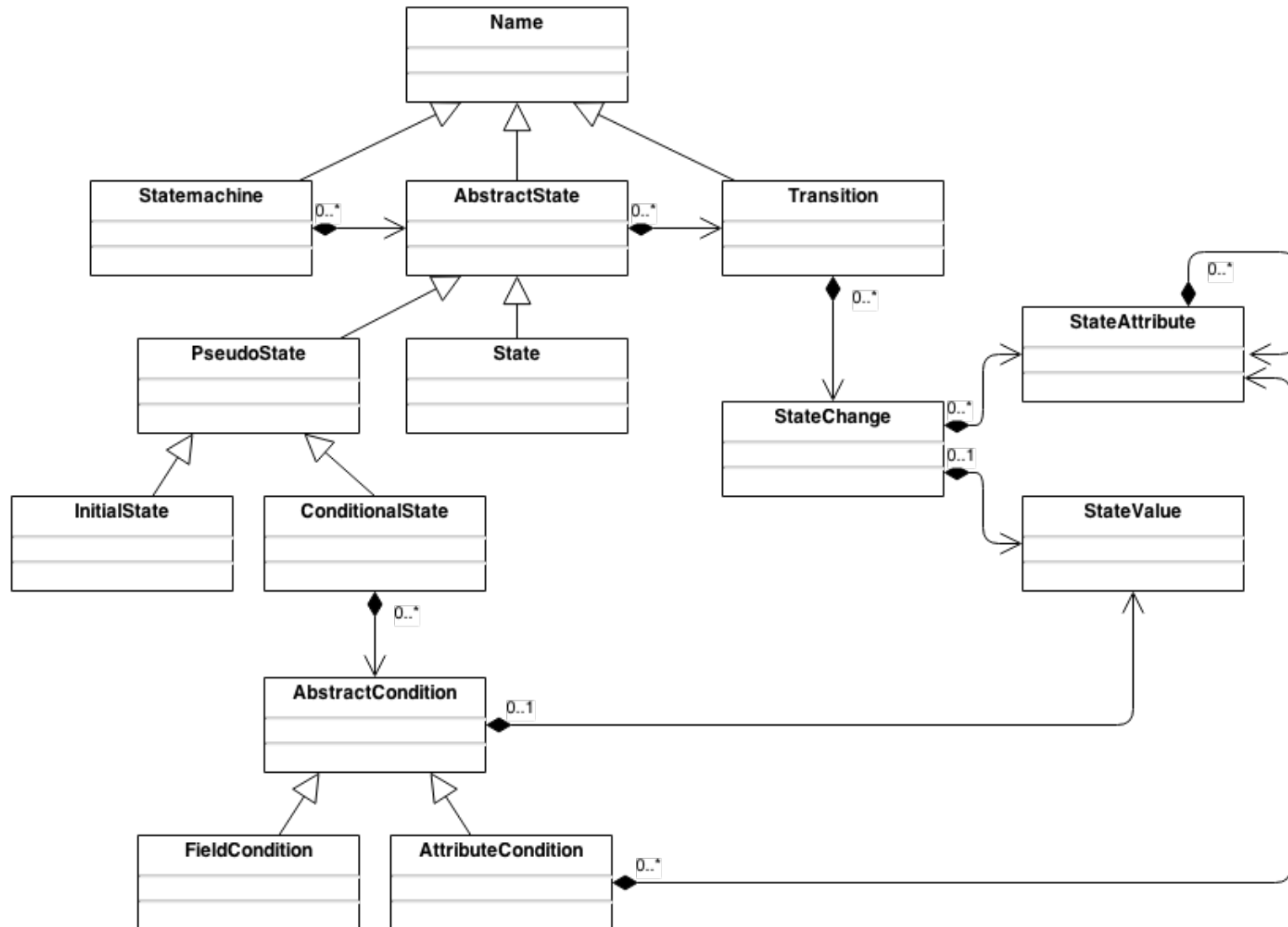
Workflow

1. Build the corresponding attribute tree for the type of trace to analyse
2. Build the state machine that represents your analysis
3. Use the generated file to execute your analysis

Results - Model

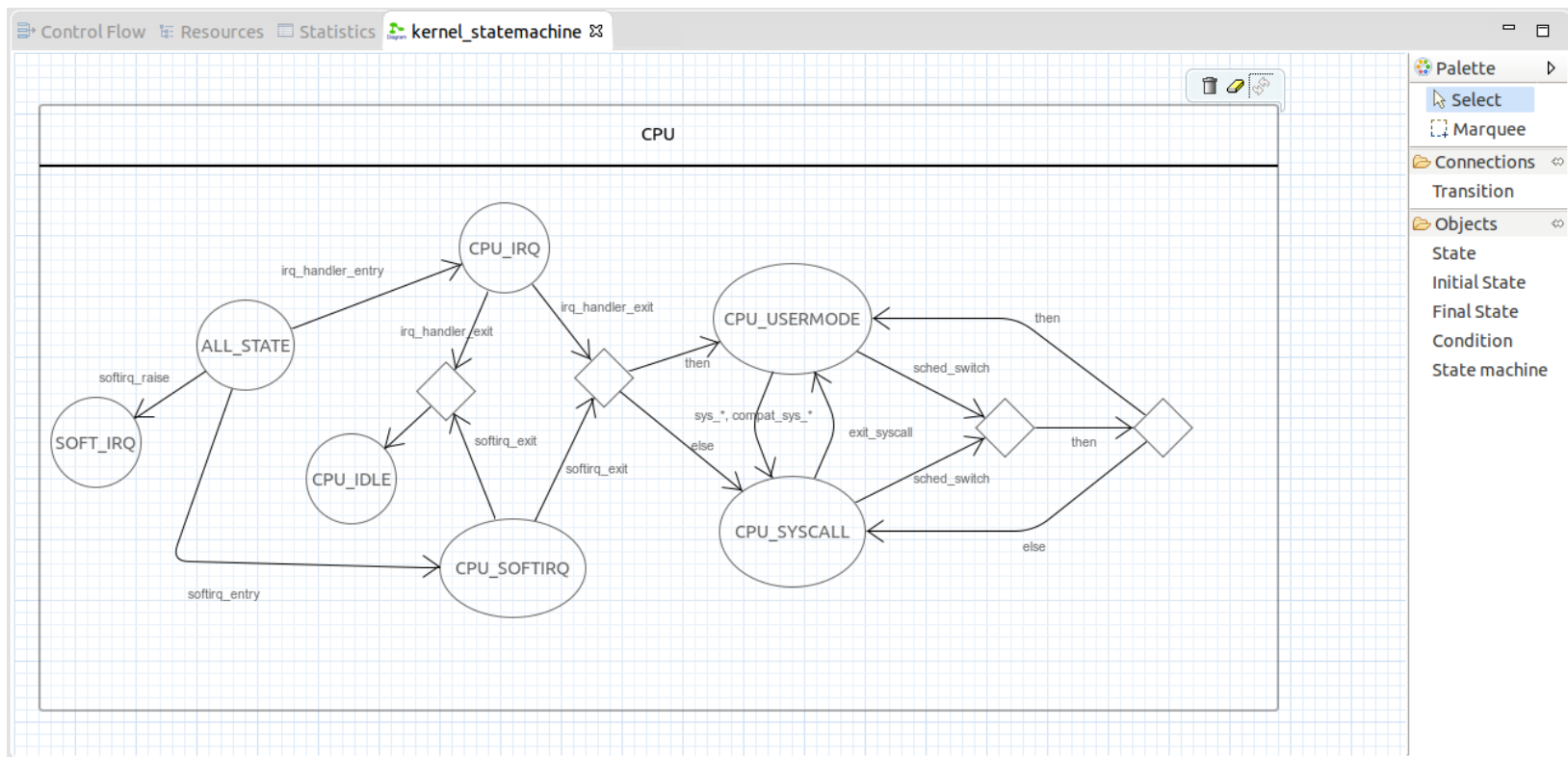
- Transition behavior -> State change
- State changes are defined as an attribute and its updated value
- The “Choice” pseudostate from UML -> Condition

Results - Model



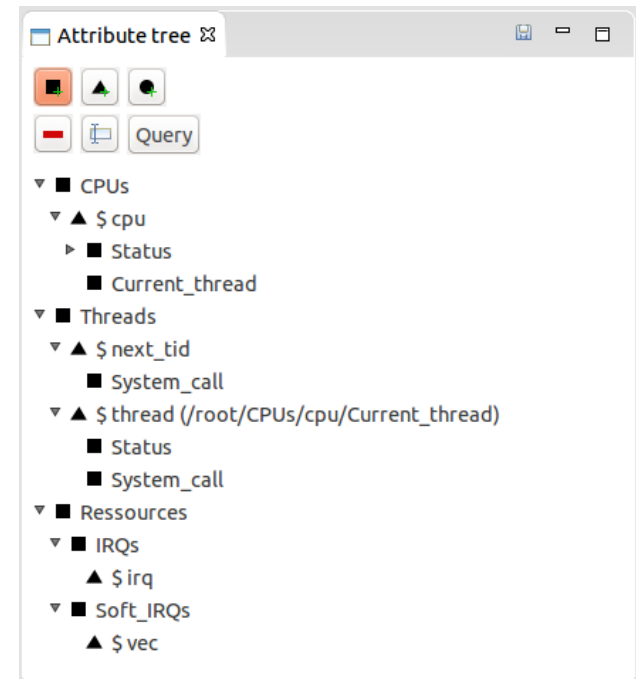
Results - Graphiti editor

- Trace analysis of the Linux kernel
- Easy to build and intuitive



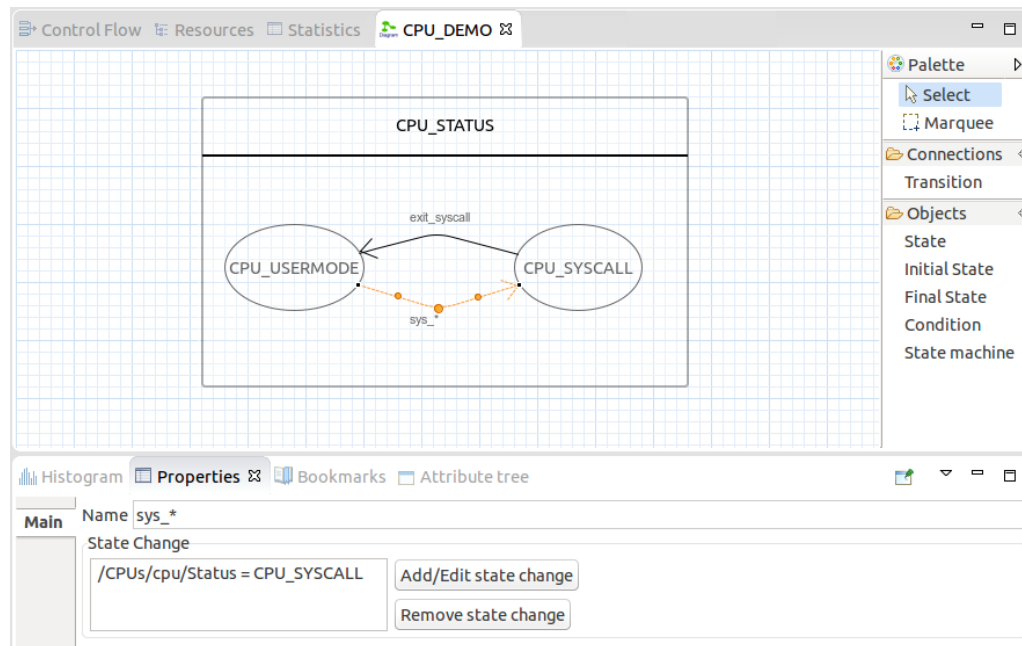
Results - Attribute tree

- Define your attribute tree once beforehand
- Simply select the defined attributes when building the state machine



Results - State machine

- Specify the attribute that will be changed with your state machine
- Automatic state change



Results - State machine

- Add additional information

The screenshot displays a state machine analysis tool interface. The main window shows a state transition diagram for a state machine named 'CPU_STATUS'. The diagram has two states: 'CPU_USERMODE' and 'CPU_SYSCALL'. There are two transitions: one labeled 'exit_syscall' from 'CPU_USERMODE' to 'CPU_SYSCALL', and another labeled 'sys_*' from 'CPU_USERMODE' to 'CPU_SYSCALL'. Below the diagram, there are tabs for 'Histogram', 'Properties', 'Bookmarks', and 'Attribute tree'. The 'Properties' tab is active, showing the name 'exit_syscall' and the state change '/CPUs/cpu/Status = CPU_USERMODE'. There are buttons for 'Add/Edit state change' and 'Remove state change'. A 'State change' dialog box is open on the right, showing a tree view of state attributes. The 'System_call' attribute is selected. The dialog also has a 'State value' section with a 'Type' dropdown set to 'eventName' and a 'Value' input field. 'OK' and 'Cancel' buttons are at the bottom of the dialog.

Control Flow Resources Statistics kernel_statemachine test CPU_DEMO

CPU_STATUS

CPU_USERMODE CPU_SYSCALL

exit_syscall

sys_*

State change

State attribute

- CPUs
 - \$ cpu
 - Status
 - Current_thread
- Threads
 - \$ thread (/root/CPUs/cpu/Current_thread)
 - Status
 - System_call
- \$ next_tid
 - System_call
- Ressources
 - IRqs
 - \$ irq
 - Soft_IRqs
 - \$ vec

State value

Type eventName

Value

OK Cancel

Properties

Name exit_syscall

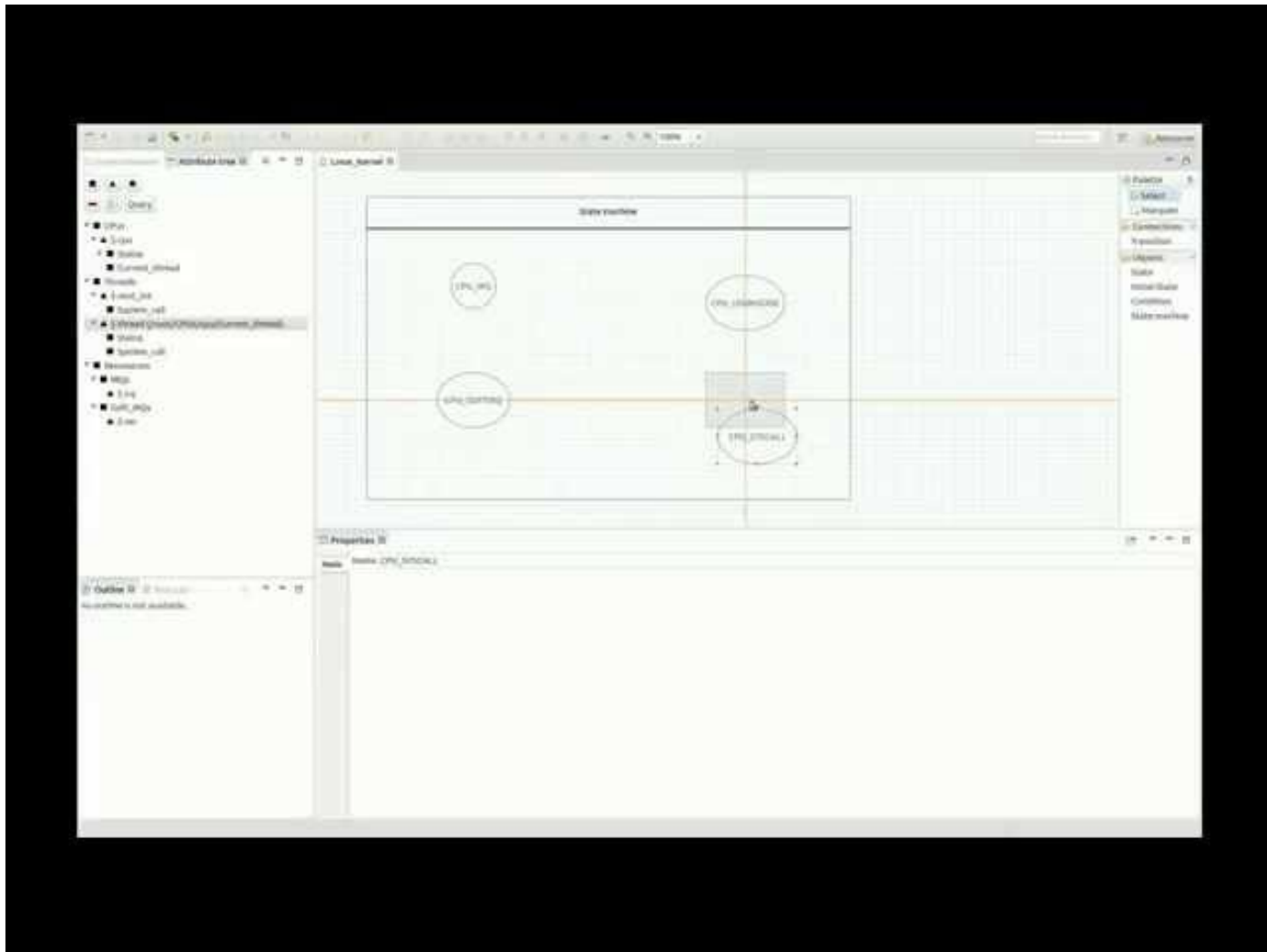
State Change

/CPUs/cpu/Status = CPU_USERMODE

Add/Edit state change

Remove state change

Demo



Results - Extract information

- We need to convert the diagram to the actual XML
- Extract information from the model that is generated
- Organize it and write the XML

Road ahead

- Filter and pattern support
- Specify views by adding information on the state diagram
- Better integration with Trace Compass
 - Synchronise views with the editor and vice versa

Conclusion

- We have an editor to capture all the information related to trace analysis
- A more efficient way to make the XML state provider