



Design of flexible analysis in TMF  
XML State-Provider and XML Filter

Presented by : Florian Winger  
DORSAL - December 10<sup>th</sup> 2013

# Introduction

---

## ▶ Objectives :

- ▶ Capture and merge observations made anywhere in the system (alerts, events, and states) by many surveillance systems (AV, HIDS, performance monitor systems, software tracers...)
- ▶ Convert this data in the most appropriate format for detection analysis.

## ▶ Proposed solution :

- ▶ Formal expressions to use the state-system in TMF
  - ▶ Allow the creation of personalized State System
  - ▶ Allow a new analysis with virtual states (filter)

# Motivation

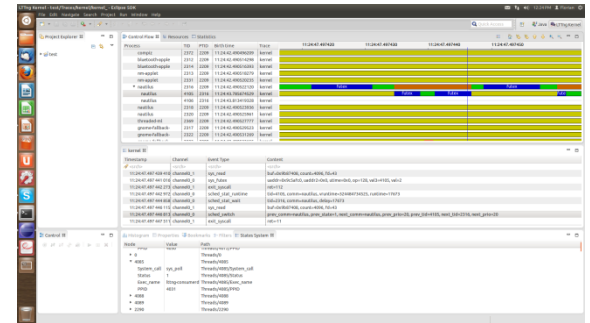
- ▶ Improve the use of the State System



CTF (Linux)  
Kernel Trace



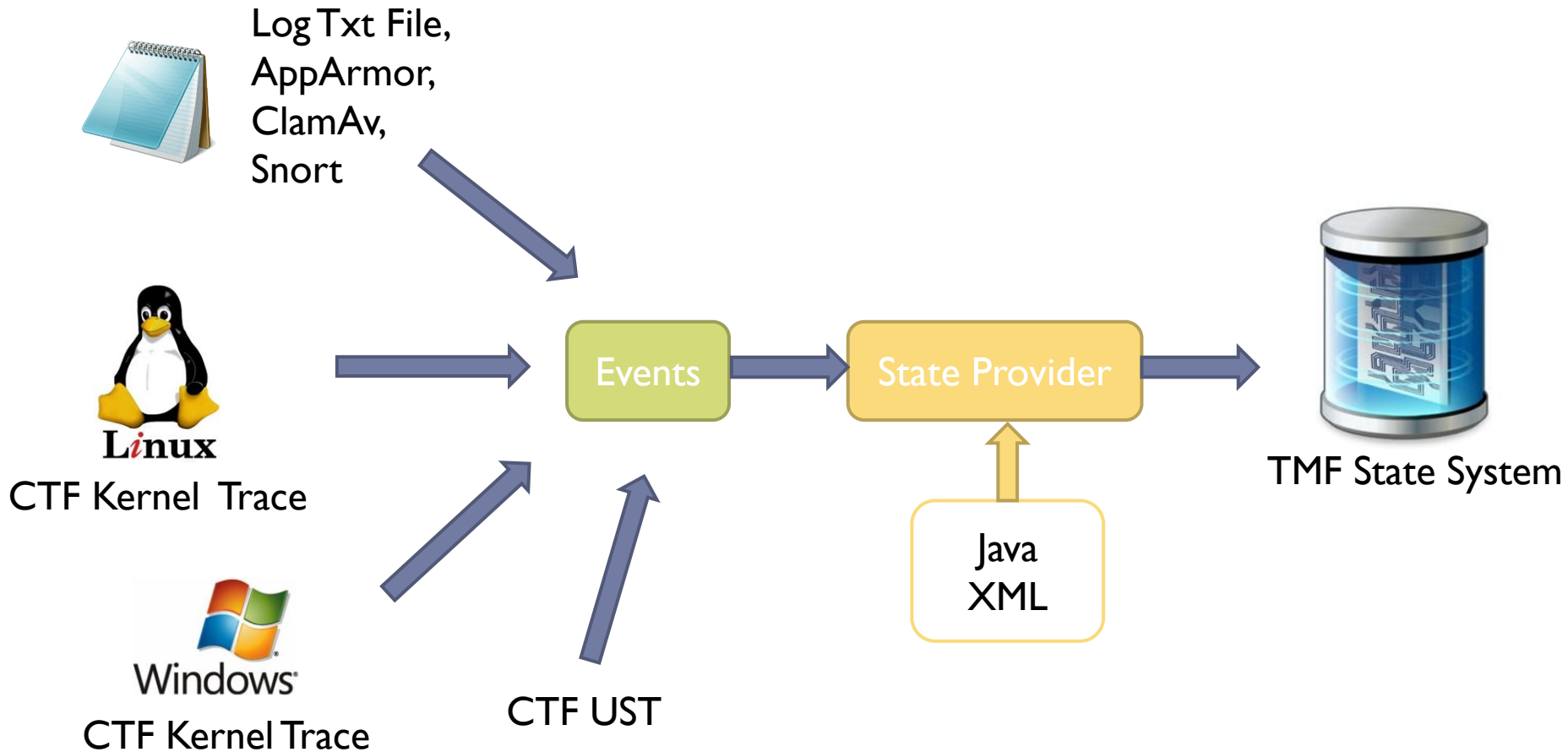
State System



TMF Control Flow View

# Motivation

---



# Motivation

---

- ▶ Currently the way to do the state system is hardcoded in a java file : `CtfKernelStateInput.java` with a big switch/case
- ▶ The idea is to transcribe it into a XML style sheet and after that we can have an interface to generate the XML.
- ▶ Multi-level information
  - ▶ We can merge multiple data's source (examples UST-Kernel, VM-Host...)

# State System Explorer

The screenshot displays the State System Explorer interface within the Eclipse IDE. The top panel shows a control flow view for various processes, including Xorg, compiz, sh, ubuntuone-syncd, gnome-terminal, gnome-pty-helpe, bash, and another bash instance. The middle panel shows a log of events such as sys\_read, exit\_syscall, and svs clock\_gettime. The bottom panel shows a hierarchical state system tree with columns for State System / Attribute, Quark, Value at timestamp, Start time, End time, and Full attribute path.

Process	TID	PTID	Birth time	Trace
Xorg	1344		11:24:42.446163619	kernel
compiz	2290		11:24:42.441770736	kernel
sh	2383	2290	11:24:42.490574292	kernel
ubuntuone-syncd	2498		11:24:42.442317455	kernel
gnome-terminal	2718		11:24:42.445170405	kernel
gnome-pty-helpe	2727	2718	11:24:42.490746730	kernel
bash	2728	2718	11:24:42.490748406	kernel
bash	3837	2718	11:24:42.490927829	kernel

Timestamp	Channel	Event Type	Content
11:24:42.442 342 039	channel0_0	sys_read	fd=3, buf=0x8e143a0, count=4096
11:24:42.442 346 648	channel0_0	exit_syscall	ret=-11
11:24:42.442 347 906	channel0_1	svs clock_gettime	which clock=1. to=0xbf6c6ab8

State System / Attribute	Quark	Value at timestamp	Start time	End time	Full attribute path
CPUs	0		11:24:42.440 133 097	11:24:52.664 579 801	CPUs
1	1		11:24:42.440 133 097	11:24:52.664 579 801	CPUs/1
0	16		11:24:42.440 133 097	11:24:52.664 579 801	CPUs/0
Current_thread	17	2290	11:24:42.442 235 251	11:24:42.442 407 549	CPUs/0/Current_thread
Status	25	2	11:24:42.442 342 039	11:24:42.442 346 647	CPUs/0/Status
Threads	3		11:24:42.440 133 097	11:24:52.664 579 801	Threads
-1	4		11:24:42.440 133 097	11:24:52.664 579 801	Threads/-1
4084	8		11:24:42.440 133 097	11:24:52.664 579 801	Threads/4084
4087	9		11:24:42.440 133 097	11:24:52.664 579 801	Threads/4087
4072	18		11:24:42.440 133 097	11:24:52.664 579 801	Threads/4072
0	19		11:24:42.440 133 097	11:24:52.664 579 801	Threads/0
4085	28		11:24:42.440 133 097	11:24:52.664 579 801	Threads/4085
4088	33		11:24:42.440 133 097	11:24:52.664 579 801	Threads/4088
4089	38		11:24:42.440 133 097	11:24:52.664 579 801	Threads/4089
2290	48		11:24:42.440 133 097	11:24:52.664 579 801	Threads/2290
Status	49	3	11:24:42.442 342 039	11:24:42.442 346 647	Threads/2290/Status
System_call	58	sys_read	11:24:42.442 342 039	11:24:42.442 346 647	Threads/2290/System_call
Exec_name	59	compiz	11:24:42.441 770 736	11:24:52.664 579 801	Threads/2290/Exec_name
PPID	60		11:24:42.440 133 097	11:24:42.490 492 646	Threads/2290/PPID

# Very interesting use case

---

## ▶ Windows Kernel Trace

- ▶ ETW : Event Trace for Windows
- ▶ .etl : etl format
- ▶ Can be use for Kernel and UST !



## ▶ Google want to optimize Chromium

- ▶ Convert ETW trace to CTF



Chromium  
Google

## ▶ Windows Performance Toolkit

# Work – Chromium project

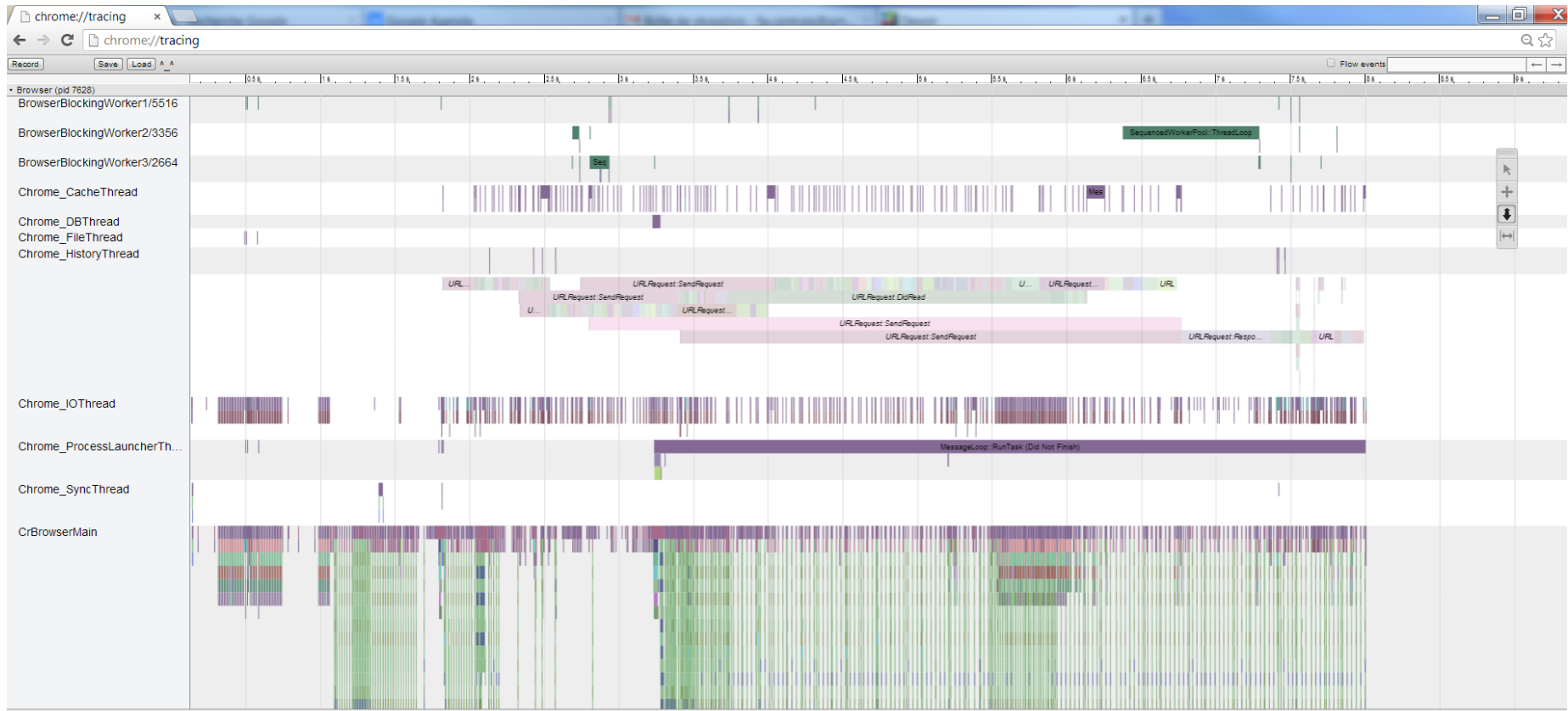
---

- ▶ **ETW2CTF convertor**
  - ▶ Allow to convert Windows Trace Format in CTF
  - ▶ <https://github.com/fwininger/ETW2CTF/>
- ▶ **Patch in Chromium**
  - ▶ Use ETW to record the existing trace points
  - ▶ Currently only for windows, but in the future, we can use LTTng-UST to extract the same information for Linux
- ▶ **Thanks to Etienne Bergeron and François Doray**



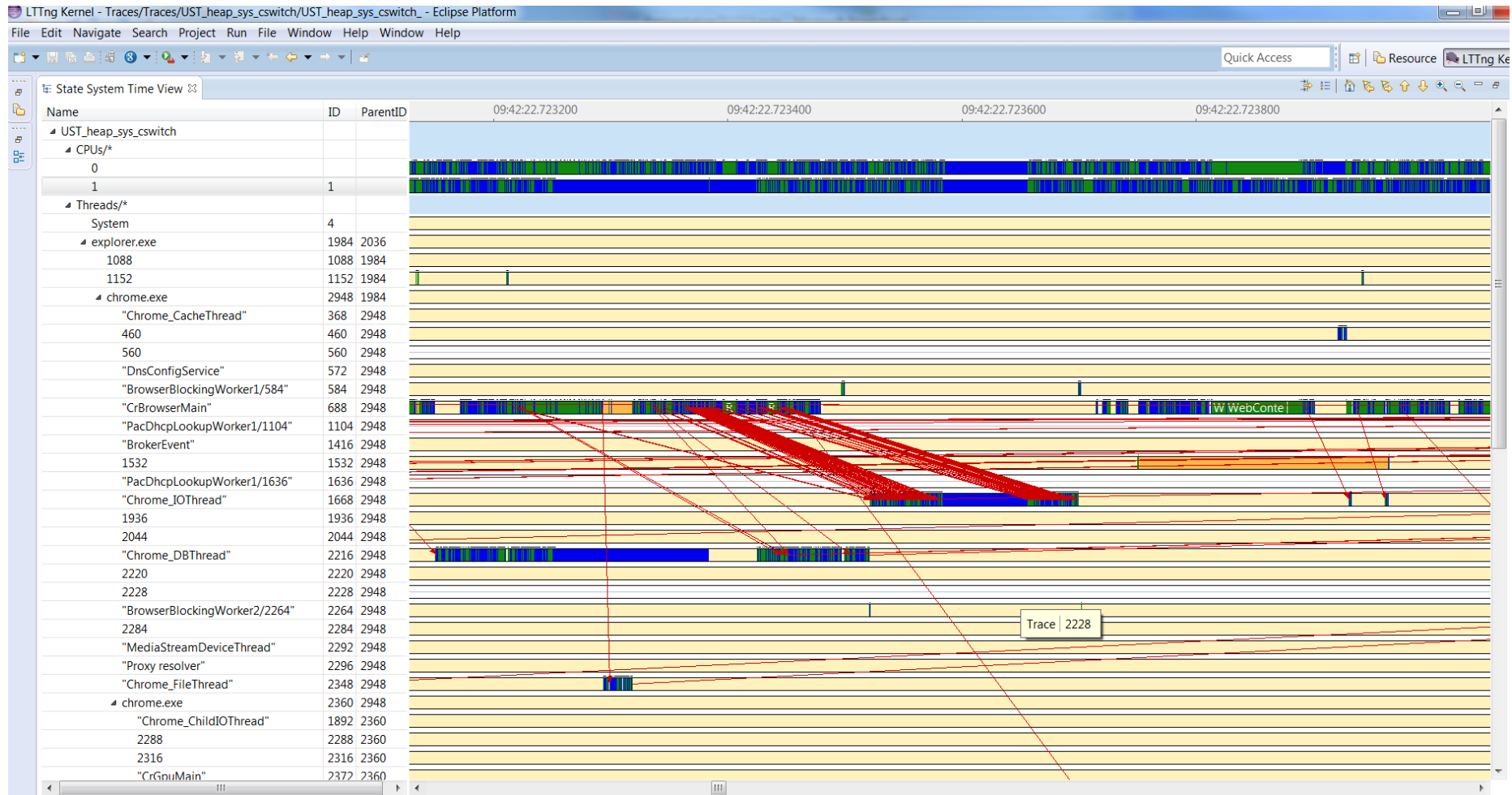
# Work – Chromium project

## ► Chrome://tracing



Selected slice:  
Title "URLRequest:SendRequest"  
Category "net"  
Start "2798.561 ms"  
Duration "3963.182 ms"  
Args  
step "SendRequest"

# Work – Chromium project



# Work – Chromium project - Advantage

---

## ▶ More information :

- ▶ Multi-level : UST and Kernel!
- ▶ Example : System Call, Wait for CPU, ...
- ▶ Memory event (alloc and free)
  - ▶ Prototype of statistics of cumulative allocation and free in a time range.

## ▶ Better scalability

- ▶ Chrome://tracing use JSON and Java Script, in memory solution

# Work – Chromium project - Limitations

---

- ▶ Currently the complete toolchain to capture and view the trace is slow, but we are working hard to change that!
  - ▶ Produce 50 MB/s of events
  - ▶ Conversion ETW to CTF : 4-5 MB/s (but can be better if we use less the WPT API)
  - ▶ Build of the state system in TMF : 2-3 MB/s

# Work – Chromium project - Limitations

---

- ▶ **Scalability problems for the UI !**
  - ▶ The State System scales to big traces, but not how it is currently used by the UI.
  - ▶ Cache added [last week] : 30x acceleration
  - ▶ Add a new query type, better suited for backend performance (automatic range query) [in review]
  - ▶ Avoid long tree branches with single child: 10x acceleration for queries when there is a large number of attributes in the current state [in review]

# Work – Make your own State System

---

- ▶ Example : UST trace (tracepoint added in GDB)

- ▶ <https://github.com/fwininger/XML4TMF>

- ▶ We want to know when the GDB is running

- ▶ 2 states :

```
<stateValue name="INF_RUNNING" value="1" />
```

```
<stateValue name="INF_STOPPED" value="0" />
```

# Work – Make your own State System

---

## ▶ I location :

```
<location id="GDB_Status">  
  <attribute constant="gdb" />  
  <attribute eventfield="pid" />  
  <attribute constant="Status" />  
</location>
```

## ▶ Example of state change :

```
<eventHandler eventname="gdb:inf_forked">  
  <stateChange>  
    <attribute location="GDB_Status" />  
    <value int="$INF_RUNNING" />  
  </stateChange>  
</eventHandler>
```

# Work – Make your own State System

```
19 <view id="org.eclipse.linuxtools.tmf.analysis.xml.sstimeview">
20   <head>
21     <analysis id="polymtl.gdb.ust.sp" />
22   </head>
23   <!-- StateValues -->
24   <stateValue name="INF_RUNNING" value="1" color="#118811" />
25   <stateValue name="INF_STOPPED" value="0" color="#CCCC" />
26
27   <!-- Control Flow View -->
28   <line id="gdb/*" display="Status" />
29
30 </view>
31 <stateprovider analysisid="polymtl.gdb.ust.sp">
32   <head>
33     <tracetype id="org.eclipse.linuxtools.tmf.ui.type.ctf" />
34     <view id="org.eclipse.linuxtools.tmf.analysis.xml.sstimeview" />
35   </head>
36
37   <!-- StateValues -->
38   <stateValue name="INF_RUNNING" value="1" />
39   <stateValue name="INF_STOPPED" value="0" />
40
41   <location id="GDB_Status">
42     <attribute constant="gdb" />
43     <attribute eventfield="pid" />
44     <attribute constant="Status" />
45   </location>
46
47   <eventHandler eventName="gdb:inf_forked">
48     <stateChange>
49       <attribute location="GDB_Status" />
50       <value int="$INF_RUNNING" />
51     </stateChange>
52   </eventHandler>
53   <eventHandler eventName="gdb:inf_stop">
54     <stateChange>
55       <attribute location="GDB_Status" />
56       <value int="$INF_STOPPED" />
57     </stateChange>
58   </eventHandler>
59   <eventHandler eventName="gdb:inf_cont">
60     <stateChange>
61       <attribute location="GDB_Status" />
62       <value int="$INF_RUNNING" />
63     </stateChange>
64   </eventHandler>
```

View's parameters

Analysis parameters

State system  
definition



# Work – Make your own State System

The screenshot displays the Eclipse IDE interface for the LTTng Kernel. The main window is titled "LTTng Kernel - GDB/Traces/64-bit/64-bit - Eclipse Platform". The interface is divided into several panes:

- Project Explorer:** Shows a tree view of the project structure, including "GDB", "Experiments [0]", "Traces [1]", and "64-bit". Under "64-bit", there is a sub-project "polymtl.gdb.ust.sp" containing "State System Explorer" and "State System Time View".
- Control Flow / Resources / State System Time View:** A table showing execution time for various components. The table has columns for time (e.g., 2013 août 13, 13:50:37.282600) and rows for components like "gdb/\*" with line numbers (2735 to 2756). Some rows are highlighted in green.
- Traces [5]:** A table listing trace events. The table has columns for "Source", "Type", "File", and "Content".
- State System Explorer:** A table showing the state system structure. The table has columns for "State System / Attribute", "Quark", "Value at timestamp", "Type", "Start time", "End time", and "Full attribute path".

The "Traces [5]" table contains the following data:

Source	Type	File	Content
<srch>	<srch>	<srch>	<srch>
37.114 920 730 4	gdb:inf_exit	channel0_4	pid=2741, filename=../gdb/gdb/inf-pttrace.c, line=179
37.282 185 697 4	gdb:cmd_run	channel0_4	filename=../gdb/gdb/infcmd.c, line=506
13:50:37.282 671 260 4	gdb:inf_for...	channel0_4	pid=2743, filename=../gdb/gdb/fork-child.c, line=384
13:50:37.282 711 381 4	gdb:inf_stop	channel0_4	pid=2743, filename=../gdb/gdb/linux-nat.c, line=3810
13:50:37.282 761 884 4	gdb:inf_cont	channel0_4	pid=2743, filename=../gdb/gdb/inf-pttrace.c, line=380
13:50:37.650 222 147 4	gdb:cmd_run	channel0_4	filename=../gdb/gdb/infcmd.c, line=506

The "State System Explorer" table contains the following data:

State System / Attribute	Quark	Value at timestamp	Type	Start time	End time	Full attribute path
64-bit						
org.eclipse.linuxtools.tmf.statistics						
total	0	223	Int	13:50:37...	13:50:37...	total
event_types	1			13:50:34...	13:51:46...	event_types
gdb:inf_forked	3	4	Int	13:50:37...	13:50:37...	event_types/gdb:...
polymtl.gdb.ust.sp						
gdb	0			13:50:34...	13:51:46...	gdb
2743	7			13:50:34...	13:51:46...	gdb/2743
Status	8	1	Int	13:50:37...	13:50:37...	gdb/2743/Status

# Work - Filtering

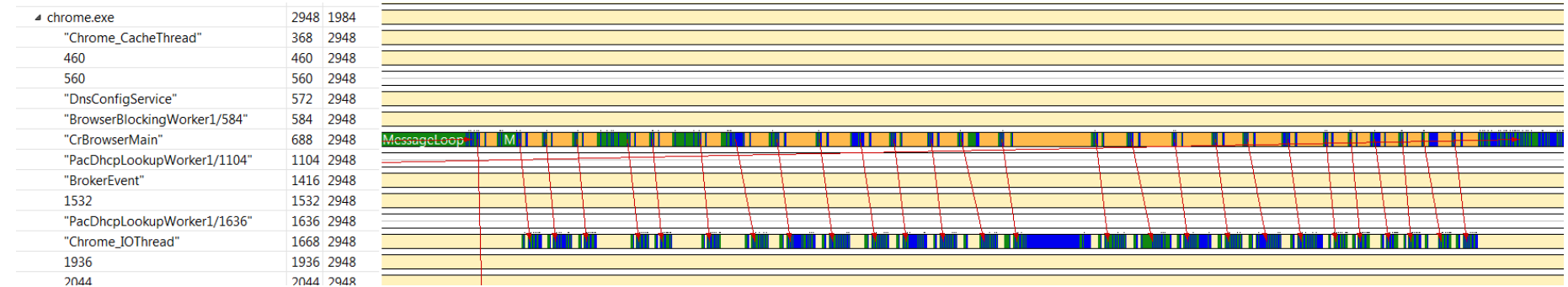
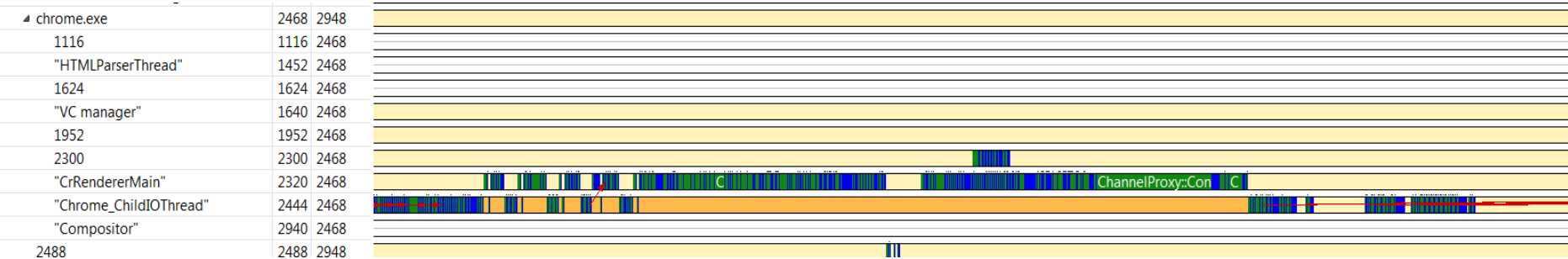
- ▶ It's very easy to extend the XML definition to make filters (based on finite state machine)

```
<filter name="wait">
  <transition start=0 end=1>
    <if>
      <condition>
        <attribute localion="Chrome" />
        <value int="$PROCESS_STATUS_WAIT_FOR_CPU" />
      </condition>
    </if>
  </transition>
  <transition start=1 end=0>
    <if>
      <condition>
        <not>
          <attribute localion="Chrome" />
          <value int="$PROCESS_STATUS_WAIT_FOR_CPU" />
        </not>
      </condition>
    </if>
  </transition>
</filter>
```



# Work - Filtering

## ▶ Example : Detecting bad Wait for CPU



# Work - Filtering

---

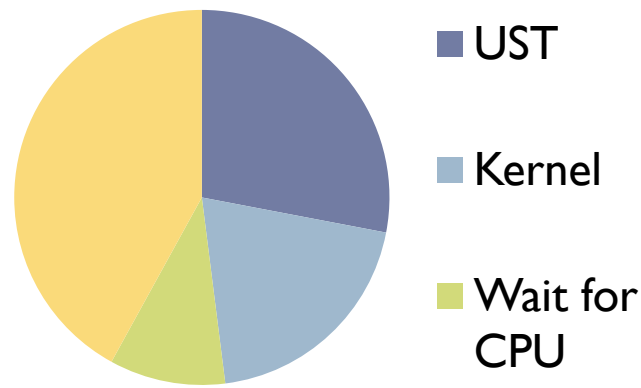
- ▶ We add a level of abstraction
- ▶ We can do the same filter based on event but it's more complex.
  - ▶ Cswitch event with the event field `OldThreadWaitReason=32` means that the thread is preempted.
- ▶ Easy way to add a trigger to detect problems.

# Work – Filtering

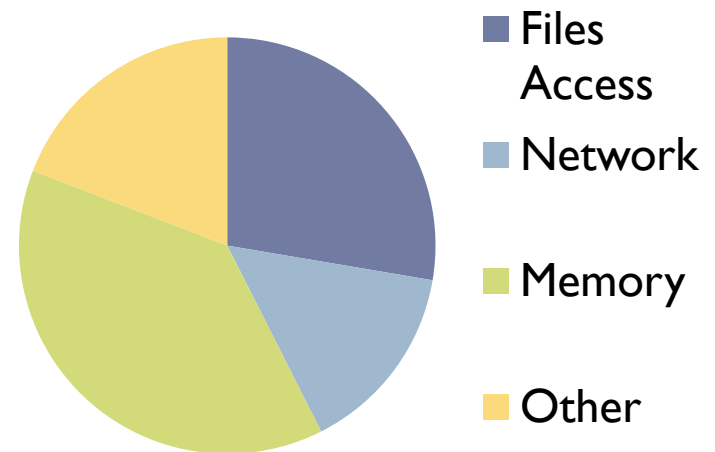
---

- ▶ In fact we add virtual state interval with the filter.
- ▶ We can use these filters to add statistics. (based on time or frequency...)

## Usage of a Thread



## Kernel Mode



# Work - Filtering

LTtng Kernel - test/Traces/kernel/kernel\_ - Eclipse SDK

File Edit Navigate Search Project Run Window Help

Project Explorer

- test
  - Experiments [0]
  - Traces [3]
    - java-sequence-single-thread
    - kernel
      - state-trace-sample.log

Control Flow

Process	TID	PTID	Birth time	Trace
Xorg	1344		11:24:42.446163619	kernel
compiz	2290		11:24:42.441770736	kernel
sh	2383	2290	11:24:42.490574292	kernel
ubuntuone-syncd	2498		11:24:42.442317455	kernel
gnome-terminal	2718		11:24:42.445170405	kernel
gnome-pty-helpe	2727	2718	11:24:42.490746730	kernel
bash	2728	2718	11:24:42.490748406	kernel
bash	3837	2718	11:24:42.490927829	kernel

State Flow View

Timestamp Channel Event Type Content

Timestamp	Channel	Event Type	Content
11:24:42.442 342 039	channel0_0	sys_read	fd=3, buf=0x8e143a0, count=4096
11:24:42.442 346 648	channel0_0	exit_syscall	ret=-11
11:24:42.442 347 906	channel0_1	svs clock_gettime	which clock=1, to=0xbfac6ab8

State System Explorer

State System / Attribute	Quark	Value at timestamp	Start time	End time	Full attribute path
CPUs	0		11:24:42.440 133 097	11:24:52.664 579 801	CPUs
1	1		11:24:42.440 133 097	11:24:52.664 579 801	CPUs/1
0	16		11:24:42.440 133 097	11:24:52.664 579 801	CPUs/0
Current_thread	17	2290	11:24:42.442 235 251	11:24:42.442 407 549	CPUs/0/Currn
Status	25	2	11:24:42.442 342 039	11:24:42.442 346 647	CPUs/0/Stat
Threads	3		11:24:42.440 133 097	11:24:52.664 579 801	Threads
-1	4		11:24:42.440 133 097	11:24:52.664 579 801	Threads/-1
4084	8		11:24:42.440 133 097	11:24:52.664 579 801	Threads/408-
4087	9		11:24:42.440 133 097	11:24:52.664 579 801	Threads/408'
4072	18		11:24:42.440 133 097	11:24:52.664 579 801	Threads/407:
0	19		11:24:42.440 133 097	11:24:52.664 579 801	Threads/0
4085	28		11:24:42.440 133 097	11:24:52.664 579 801	Threads/408!
4088	33		11:24:42.440 133 097	11:24:52.664 579 801	Threads/408!
4089	38		11:24:42.440 133 097	11:24:52.664 579 801	Threads/408!
2290	48		11:24:42.440 133 097	11:24:52.664 579 801	Threads/229!
Status	49	3	11:24:42.442 342 039	11:24:42.442 346 647	Threads/229!
System_call	58	sys_read	11:24:42.442 342 039	11:24:42.442 346 647	Threads/229!
Exec_name	59	compiz	11:24:42.441 770 736	11:24:52.664 579 801	Threads/229!
PPID	60		11:24:42.440 133 097	11:24:42.490 492 646	Threads/229!

Usage of a Thread

Legend:

- UST
- Kernel
- Wait for CPU
- Wait

# Conclusion

---

- ▶ Formal expressions to have a more efficient use of the state-system in TMF
- ▶ Good way to easily use external data and make specific and flexible analysis
- ▶ More users, more contributors

# Questions

---

