



# Low disturbance multi-level observation and production of enhanced data

DORSAL Lab

Presented by : Florian Winger  
Supervisor : Professor Michel Dagenais

# 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

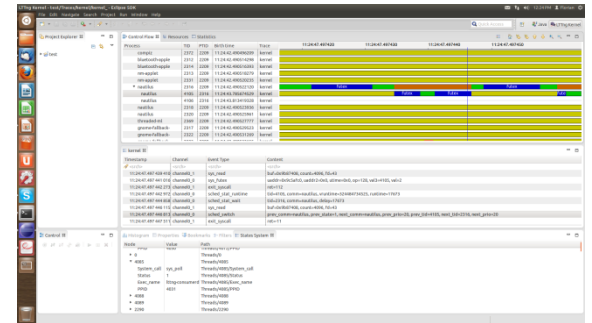
- ▶ Improve the use of the State System



CTF (Linux)  
Kernel Trace



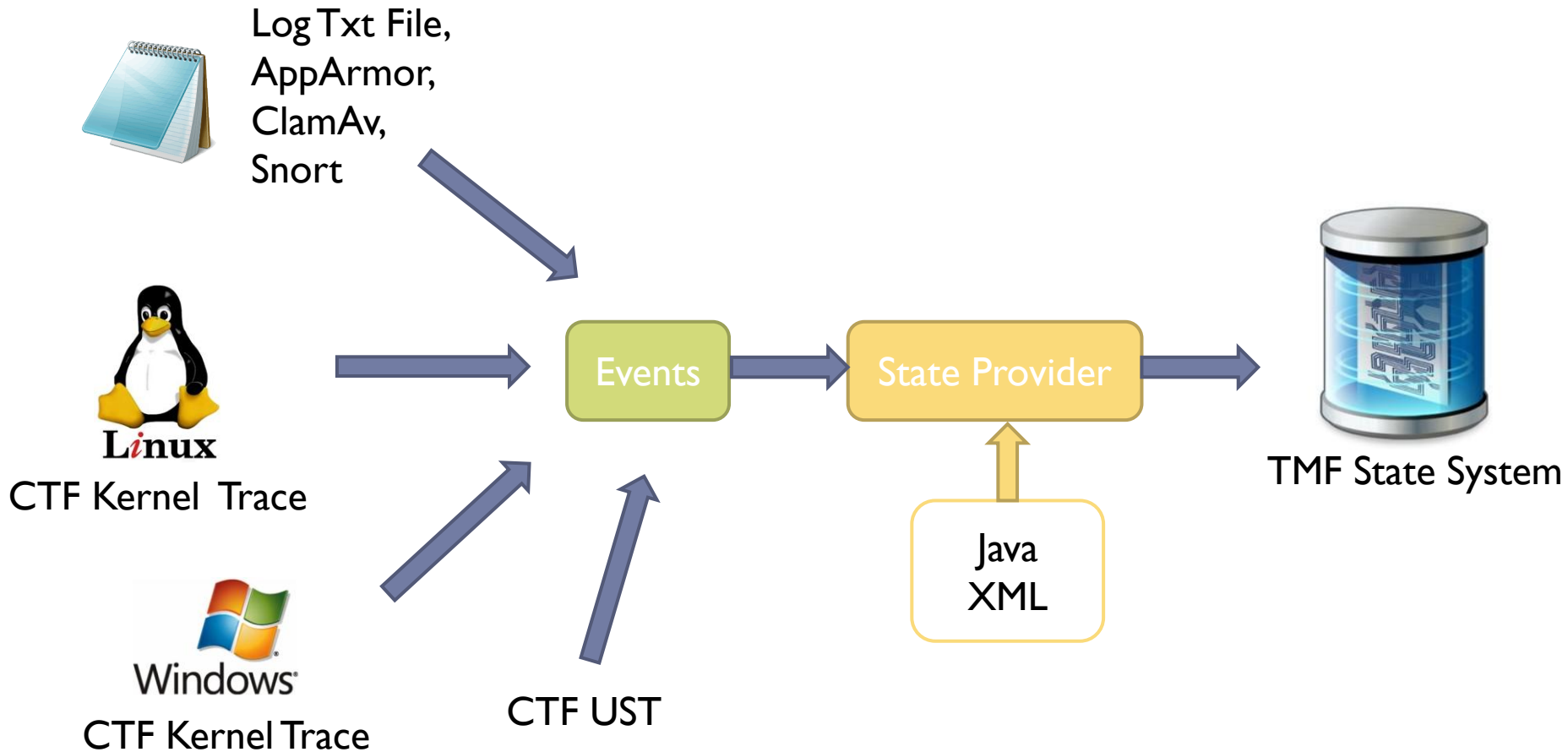
State System



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 (at first) and then in the CTF metadata with an expressive language.

# State System Explorer

The screenshot displays the State System Explorer tool within the Eclipse IDE. The main window is titled "LTng Kernel - test/Traces/kernel/kernel\_ - Eclipse SDK". The interface is divided into several panes:

- Project Explorer:** Shows the project structure, including "test", "Experiments [0]", "Traces [3]", "java-sequence-single-thread", "kernel", and "state-trace-sample.log".
- Control Flow:** A table listing processes and their state transitions.
 

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:** A Gantt chart showing the state transitions of the selected process (compiz) over time. The x-axis represents time, with markers at 11:24:42.442300, 11:24:42.442350, and 11:24:42.442400. The y-axis lists processes. The chart shows various states like "g", "c", "r", "poll", and "c" for different processes.
- Log Viewer:** Displays system events and their 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=0xbf6c6ab8
- State System Explorer:** A detailed table showing the state system hierarchy and attributes.
 

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

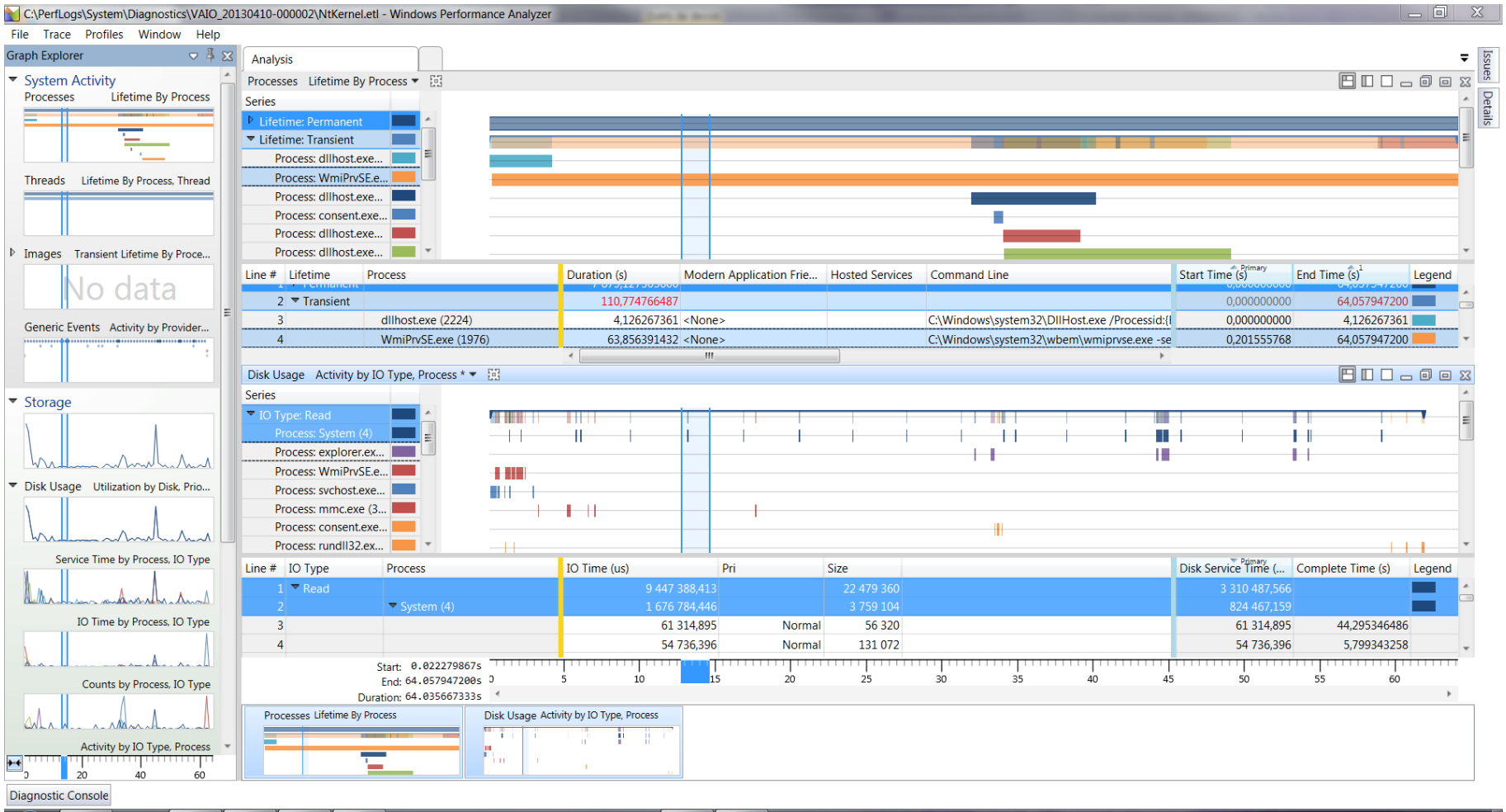
# Motivation

---

- ▶ **Windows Kernel Trace**
  - ▶ ETW : Event Trace for Windows
  - ▶ .etl : etl format
  
- ▶ **Google want to optimize Chrome**
  - ▶ Convert ETW trace to CTF
  
- ▶ **Windows Performance Toolkit**



# Windows Performance Toolkit





# Motivation

---

- ▶ **Automated Fault Identification,**
  - ▶ Béchir Ktari, Hashem Waly, Université de Laval
  - ▶ Automating the detection in Ittng traces of malicious behaviours, performance degradation, and software bugs.
  - ▶ They provide a descriptive language for scenarios
    - ▶ State machine
  - ▶ Source of inspiration to the second part: analysis with virtual states (filter)

# Work

---

- ▶ Expression language to model the state transitions caused by an event, and filter from the state system

*/Threads/4/state == RUNNING*

*/Threads/\${/event/tid}/files/\${/event/fd}/pathname = name*

# Work

---

## ▶ State Provider

## ▶ Java Code :

```
/* Put the process' status back to user mode */  
quark = ss.getQuarkRelativeAndAdd(currentThreadNode, Attributes.STATUS);  
value = TmfStateValue.newValueInt(StateValues.PROCESS_STATUS_RUN_USERMODE);  
ss.modifyAttribute(ts, value, quark);
```

## ▶ Expression code :

```
/* Put the process' status back to user mode */  
  
/Threads/[${/CPUs/[${/event/Cpu}]/Current_thread}]/Status  
= PROCESS_STATUS_RUN_USERMODE
```

# Work

---

- ▶ State Provider

- ▶ Expression :

```
/* Put the process' status back to user mode */  
/Threads/[${/CPUs/[${/event/Cpu}]/Current_thread}]/Status  
= PROCESS_STATUS_RUN_USERMODE
```

- ▶ XML :

```
<stateprovider>  
  <location id="CurentThread">  
    <attribute type="hardcoded" name="Threads" />  
    <attribute type="query">  
      <attribute type="hardcoded" name="CPUs" />  
      <attribute type="field" fieldValue="cpu" />  
      <attribute type="hardcoded" name="Current_thread" />  
    </attribute>  
  </location>  
  
  <eventHandler eventName="exit_syscall">  
    <stateChange type="modify">  
      <attribute type="location" id="CurentThread">  
        <attribute type="hardcoded" name="Status" />  
        <value type="int" name=" PROCESS_STATUS_RUN_USERMODE "/>  
      </stateChange>  
    </eventHandler>  
  </stateprovider>
```

# Work

## ► XML : Aaron Spear



The screenshot shows the Eclipse IDE interface with the following components:

- Project Explorer:** Shows a project named 'esx-events' containing 'Experiments [0]', 'Traces [2]', 'java-sequence-single-thread', and 'state-trace-sample.log'.
- State Trace Log Table:** A table with columns: Time Stamp, Datacenter, Host, VM, Event Type, and Message. It lists several events such as VM starting and stopping.
- State Flow View:** A diagram showing the execution flow between contexts: YourDataCenter, YourHypervisor, A VM, C VM, B VM, MyHypervisor, and another VM. Each context has a corresponding trace file 'state-trace-sample.log'.
- Time Chart:** A horizontal bar chart showing the duration of each context's execution over time.
- Histogram:** A vertical bar chart showing the frequency of events over time.

# Work

## ► XML : Aaron Spear



The screenshot shows the Eclipse IDE interface with the LTTng Kernel plugin. The main window displays a table of trace events for a Java application. The table has four columns: Timestamp, Channel, Event Type, and Content. The events are as follows:

Timestamp	Channel	Event Type	Content
15:30:42.104 576 469	channel0_1	lttng_ust_java:function_entry_event	int_tid=0x4464, string_class=com.vmware.eventanalyzer.demo.javaMethodTraceExample, string_method=<clinit>, int_lineno=24
15:30:42.104 816 408	channel0_1	lttng_ust_java:function_exit_event	int_tid=0x4464, string_class=com.vmware.eventanalyzer.demo.javaMethodTraceExample, string_method=<clinit>, int_lineno=29
15:30:42.104 915 821	channel0_1	lttng_ust_java:function_entry_event	int_tid=0x4464, string_class=com.vmware.eventanalyzer.demo.javaMethodTraceExample, string_method=<init>, int_lineno=22
15:30:42.104 930 449	channel0_1	lttng_ust_java:function_exit_event	int_tid=0x4464, string_class=com.vmware.eventanalyzer.demo.javaMethodTraceExample, string_method=<init>, int_lineno=22
15:30:42.105 891 574	channel0_1	lttng_ust_java:function_entry_event	int_tid=0x4496, string_class=com.vmware.eventanalyzer.demo.javaMethodTraceExample, string_method=run, int_lineno=34
15:30:42.108 773 745	channel0_1	lttng_ust_java:function_entry_event	int_tid=0x4496, string_class=com.vmware.eventanalyzer.demo.javaMethodTraceExample, string_method=function1, int_lineno=55
15:30:42.111 430 367	channel0_1	lttng_ust_java:function_exit_event	int_tid=0x4496, string_class=com.vmware.eventanalyzer.demo.javaMethodTraceExample, string_method=function2, int_lineno=63

Below the table, the Time Chart shows the execution flow of the application. The chart has a context menu with the following items:

Context	Trace
<init>	java-sequence-single-thread
17558	java-sequence-single-thread
Method	java-sequence-single-thread
run	java-sequence-single-thread
function1	java-sequence-single-thread
function2	java-sequence-single-thread
function3	java-sequence-single-thread

The Time Chart shows the execution of the run method, which calls function1, function2, and function3. The run method starts at 15:30:42.106 and ends at 15:30:42.124. Function1 starts at 15:30:42.108 and ends at 15:30:42.112. Function2 starts at 15:30:42.112 and ends at 15:30:42.116. Function3 starts at 15:30:42.116 and ends at 15:30:42.124.

At the bottom, the Histogram shows the distribution of events. The current event is at 15:30:42.104 576 469. The window span is 000.100 000 000. The histogram shows a single event at this timestamp.

# Work

---

## ▶ Context (view parameter)

```
<defineContext id="Process" parentId="" hasState="true"/>
<defineContext id="Thread" parentId="Process" hasState="true"/>
<defineContext id="Method" parentId="Thread" hasState="true"/>
```

## ▶ State Declaration

```
<stateDeclaration>
  <state name="Running" rgb="23,207,93">
    <value>I</value>
  </state>
</stateDeclaration>
```

## ▶ Switch Context (State Change)

```
<switchContext eventType="ltnng_ust_java:function_entry_event">
  <valueContext id="Thread" field="int_tid" fieldRegex="(.*)" />
</switchContext>
```

# Next Move

---

- ▶ Validate the syntax (expressiveness!)
- ▶ Make the Kernel State Provider for Windows with it
  - ▶ Analyze and prove it work with new architecture
- ▶ Test performance compared to Java code
- ▶ Convert the syntax to use with the CTF Metadata
  - ▶ Independence of the viewer
- ▶ Make a Gui to use virtual state to have filter with the State System.



# 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

---



# Sources

---

- ▶ Automated Fault Identification, Kernel Trace Analysis, Mémoire de Hashem Waly, Université de Laval, 2011
- ▶ Windows ADK : <http://msdn.microsoft.com/en-us/performance/cc825801.aspx>
  - ▶ TraceLog
  - ▶ Windows Performance Toolkit