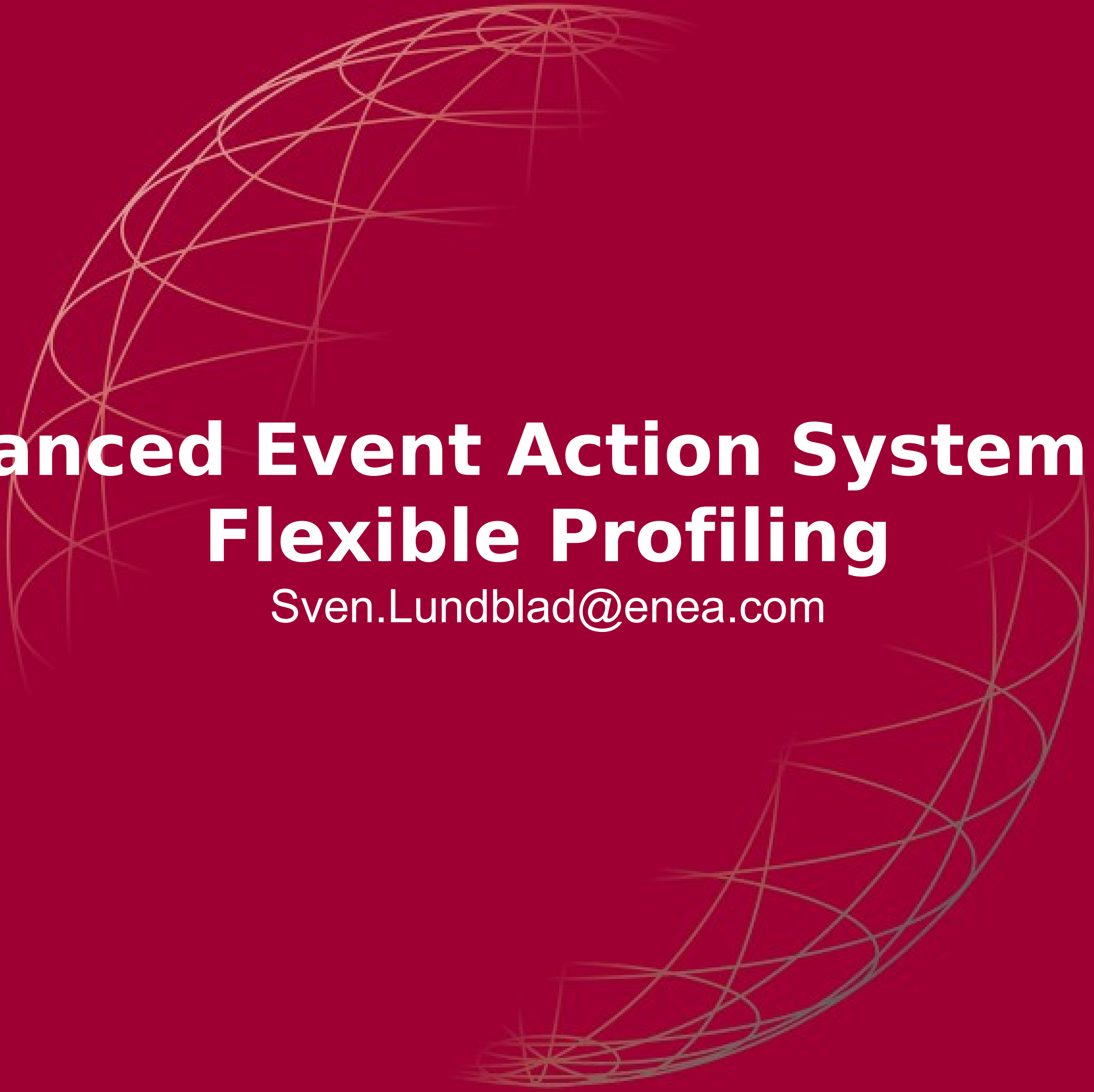
ENEA



Advanced Event Action System and Flexible Profiling

Sven.Lundblad@enea.com

Overview

- ENEA Optima and OSE Introduction
- Advanced Event Action System
- Report Profiling – A flexible profiling system
- Taking the Event and Profiling System to Linux

What the heck is OSE?

- Operating System
- Real-time Operating System
- Message Passing Real-time Operating System
- Distributed Message Passing Real-time Operating System
- Fault handling Distributed Message Passing Real-time Operating System
- Multicore Fault handling Distributed Message Passing Real-time Operating System!

OSE

- Light weight processes with resource tracking
- Simple *and* Powerful Signal API (messages)
  - Hunt, attach, alloc, send, receive, and free_buf
  - Signals are asynchronically
  - All messages can, if wanted, be received and handled from any place
- Built in supervision of peers (attach)
- Programs with optional memory protection
- Forward error recovery mechanism built in
- Micro kernel approach (but not too “micro”)

Optima

- Eclipse based tool suite for OSE, OSEck, and Linux (soon)
  - Application and system development tools (typical IDE functionality)
  - System browser
  - Profiling and analyse tools
  - Tracing and Event Action tools
  - Post mortem tools
  - Multicore support
  - Flexible target connection

Optima System Browser

Hierarchical view (System model) (Context menus)

Gateway

Target

Block

Processes

Type and state decorations

Load modules view (Context menu)

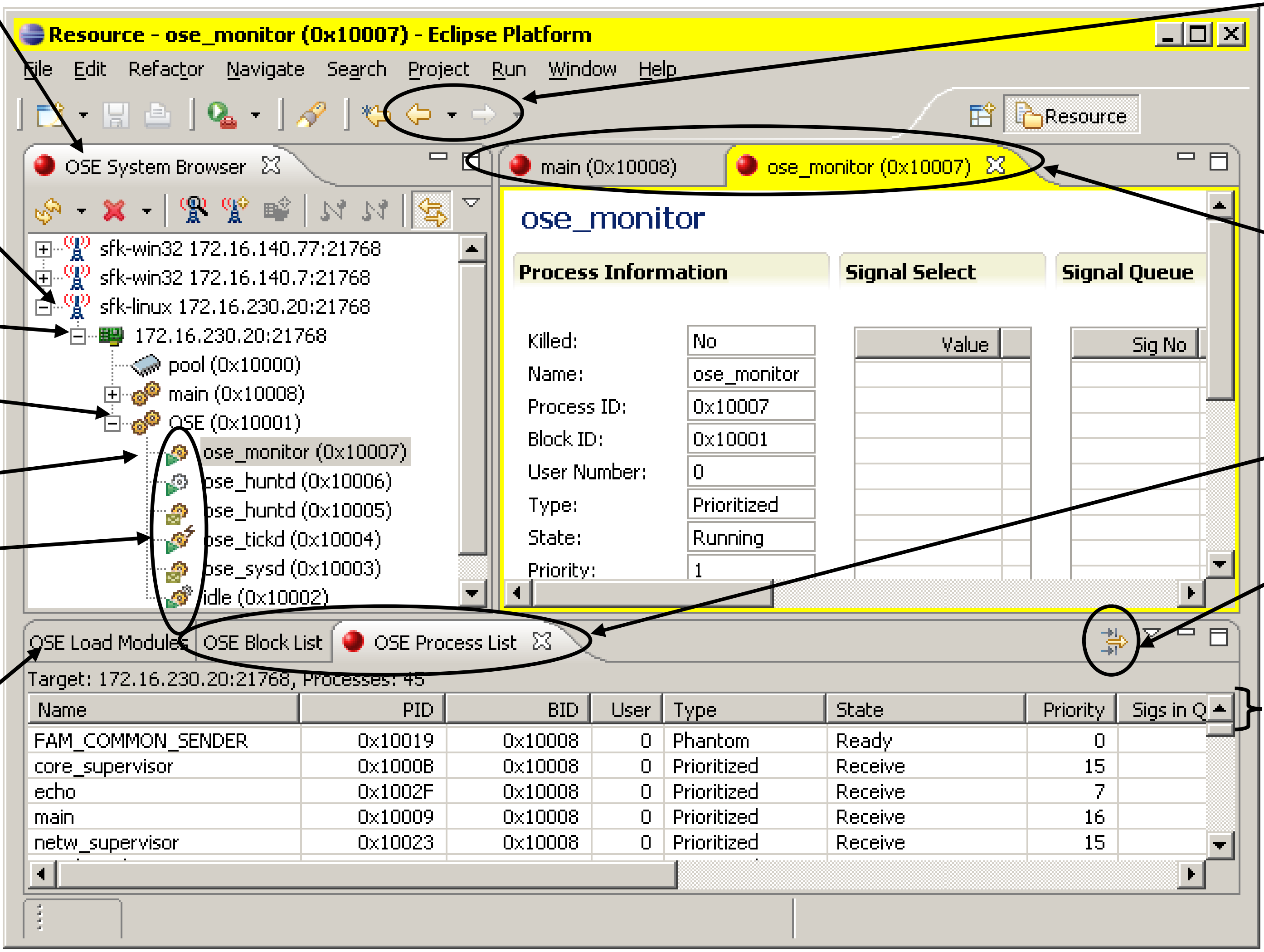
Editor navigation Back / Forward

Details editors (System model) (Double click)

Table data views (Block) (Process)

On target filter (All properties)

Sortable (All properties)



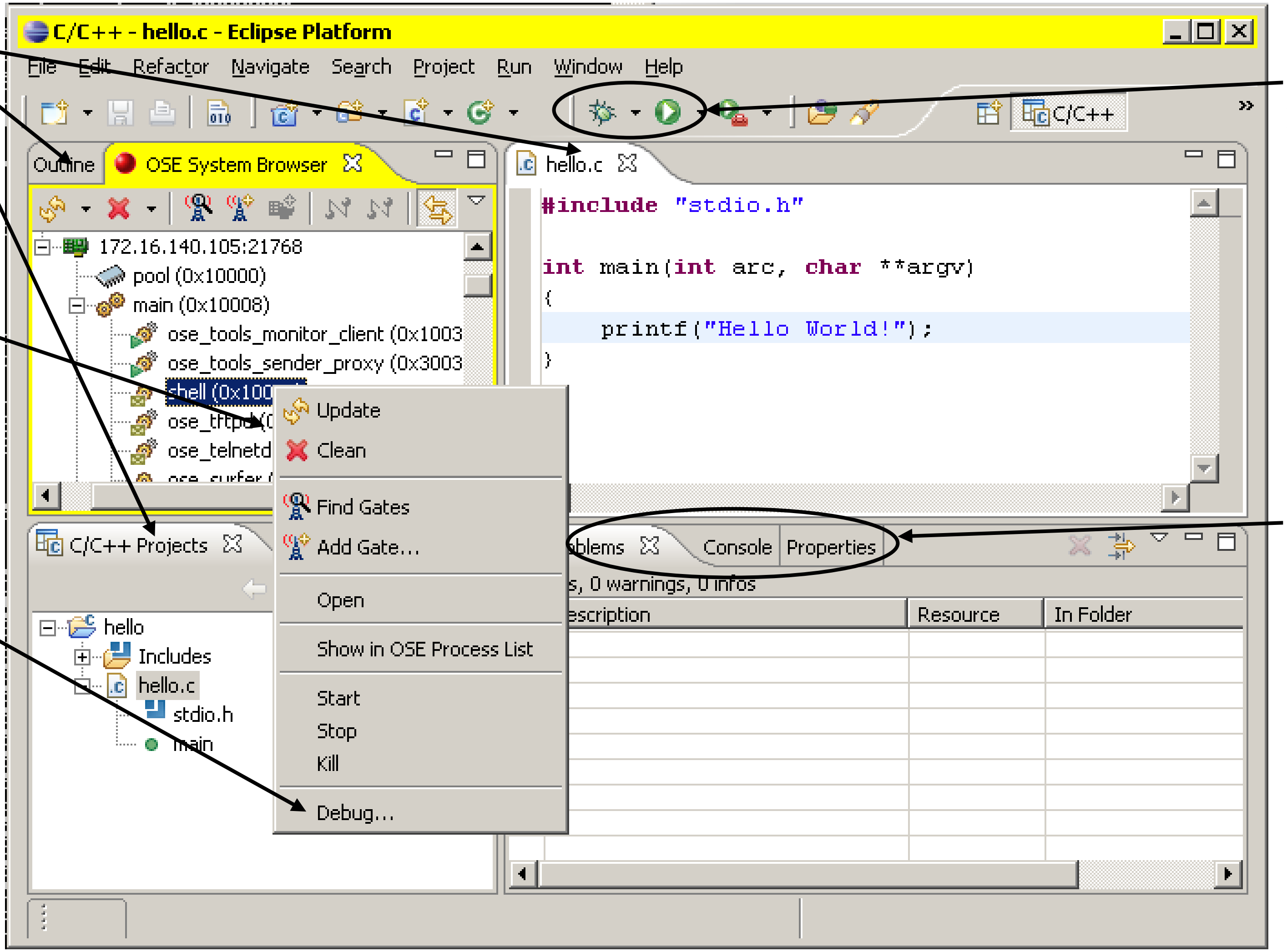
ose_monitor

Killed:	No
Name:	ose_monitor
Process ID:	0x10007
Block ID:	0x10001
User Number:	0
Type:	Prioritized
State:	Running
Priority:	1

Target: 172.16.230.20:21768, Processes: 45

Name	PID	BID	User	Type	State	Priority	Sigs in Q
FAM_COMMON_SENDER	0x10019	0x10008	0	Phantom	Ready	0	
core_supervisor	0x1000B	0x10008	0	Prioritized	Receive	15	
echo	0x1002F	0x10008	0	Prioritized	Receive	7	
main	0x10009	0x10008	0	Prioritized	Receive	16	
netw_supervisor	0x10023	0x10008	0	Prioritized	Receive	15	

Optima System Browser



```
#include "stdio.h"

int main(int arc, char **argv)
{
    printf("Hello World!");
}
```

CDT editor/views

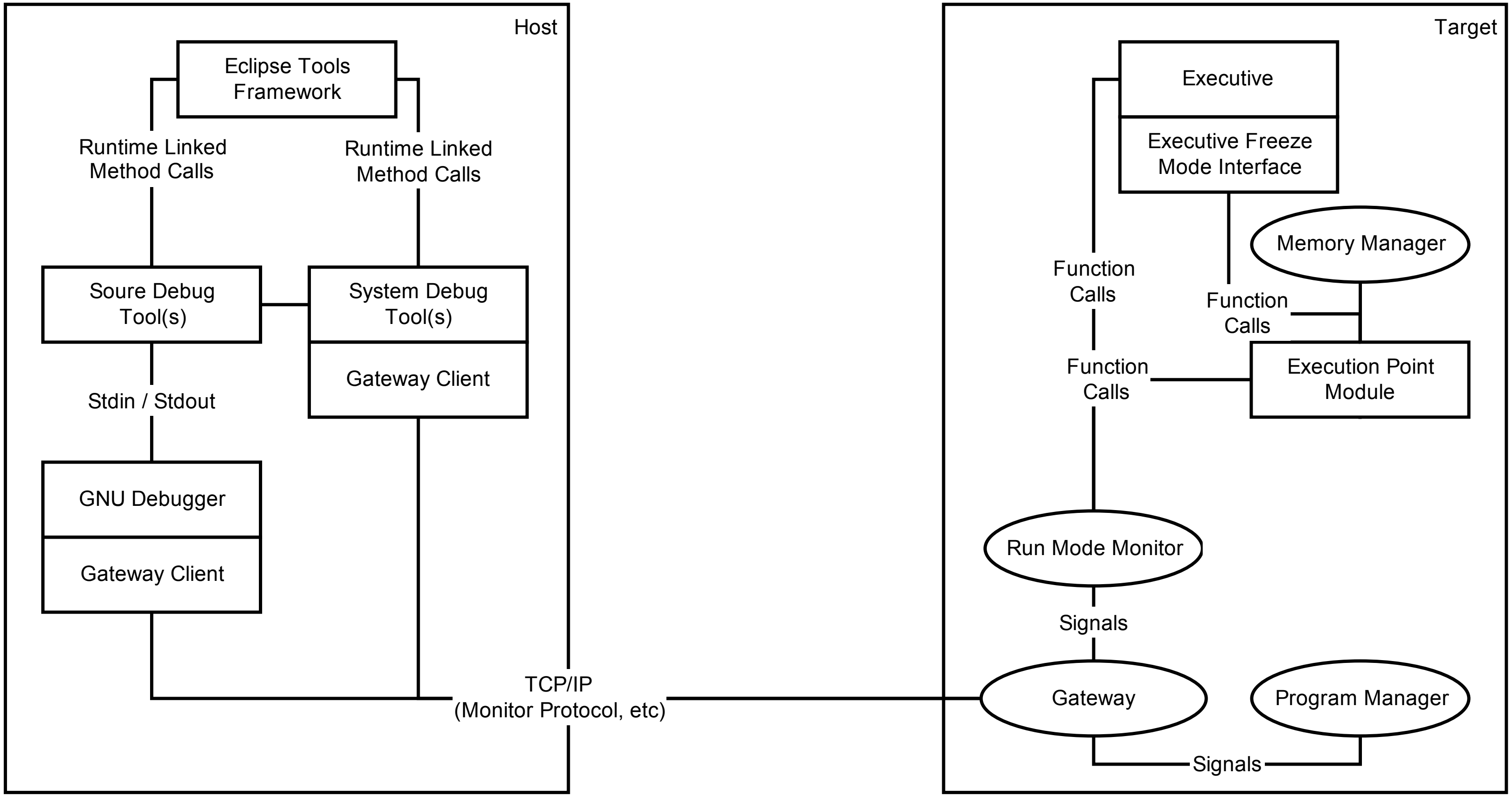
Operations (Context) (Contributions)

CDT Integration (Launch Configuration) (Two click load and launch OSE programs)

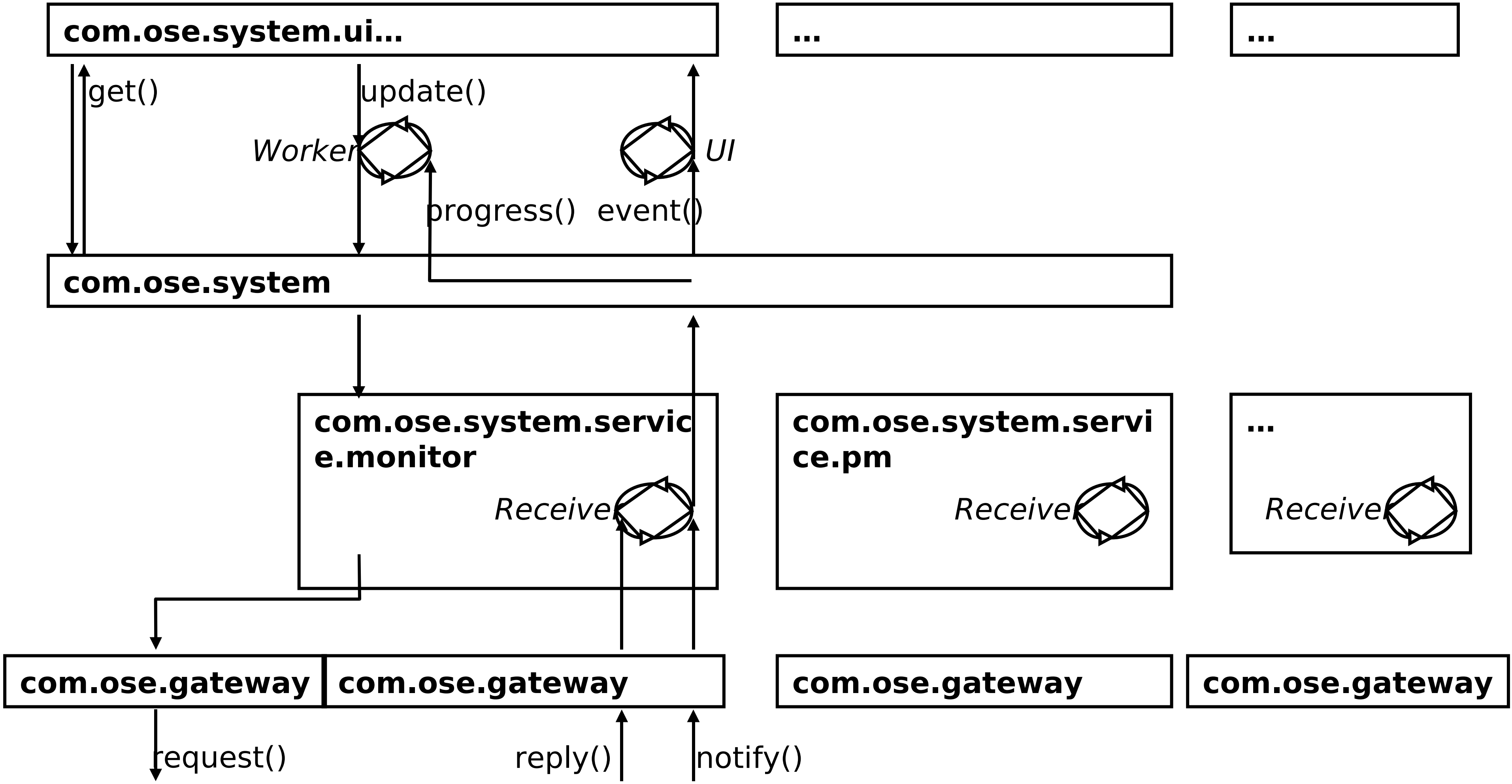
CDT Relaunch/debug

Platform views

Optima and OSE5



Optima Architecture



com.ose.system.ui…

…

…

get()

update()

Worker

UI

progress()

event()

com.ose.system

com.ose.system.servic e.monitor

Receiver

com.ose.system.servi ce.pm

Receiver

…

Receiver

com.ose.gateway

com.ose.gateway

com.ose.gateway

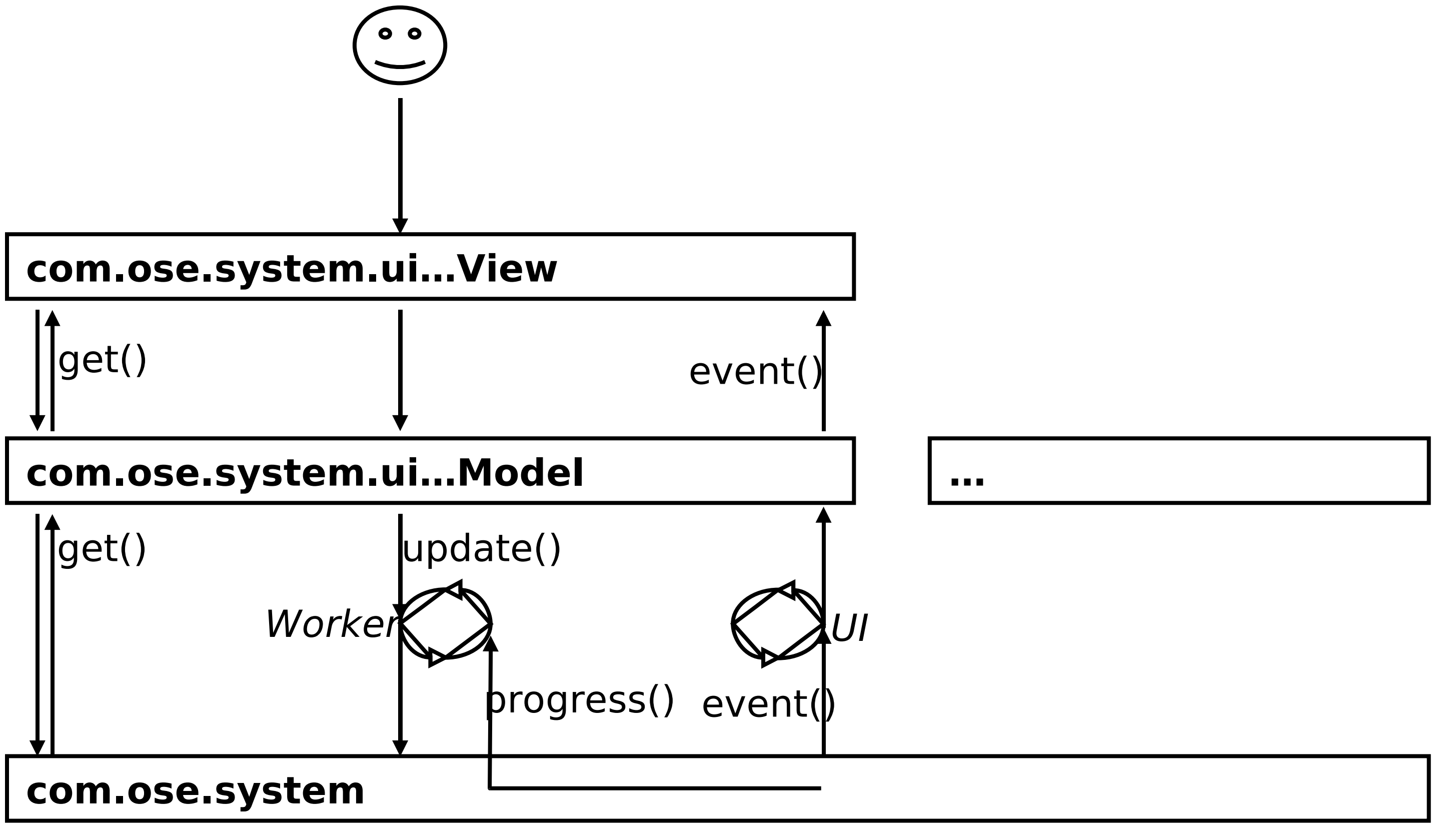
com.ose.gateway

request()

reply()

notify()

Optima Architecture



com.ose.system.ui...View

get()

event()

com.ose.system.ui...Model

...

...

get()

update()

Worker

UI

progress()

event()

com.ose.system

Advanced Event Action System

- Decoupling of events and actions
- Actionpoints defines rules that couple an event to a particular action
- An actionpoint contains event conditions for whenever it should be trigged or not
- Actionpoints is associated with a state
- Actionpoint rules are only evaluated when activated and the event is from a relevant process (a process scope)
- Because of above the event system has a very low intrusiveness when not used

Events

- Events are send, receive, create, kill, swap, error, bind, user events
- Events have meta data about
  - OS time stamp
  - Real calendar time (optional)
  - Process causing event
  - Additional event type specific information
- Event data such as signal data or text from application
- Trace data can be uploaded and displayed in Optima
- In Optima trace data can be saved in text files (XML) for further processing

Actions

- Decoupling of events and actions
- Actionpoints defines rules that couple an event to a particular action
- Actions are trace, notify, intercept, enable trace, disable trace, set state, undo event, user action

User Events

- Applications can report events with or without data (of variable size)
- An unique identifier describes the event type (similar to OSE signal numbers)
- The event data is described with a C-struct (could be the applications native data structure)
- A simple API with only two functions implemented by the OS
- Almost non intrusive when not used
- Application events coordinated with system events in the same trace
- Optima can automatically view the event data with symbolic information (based on the SigDB tool)

User Event Use Cases

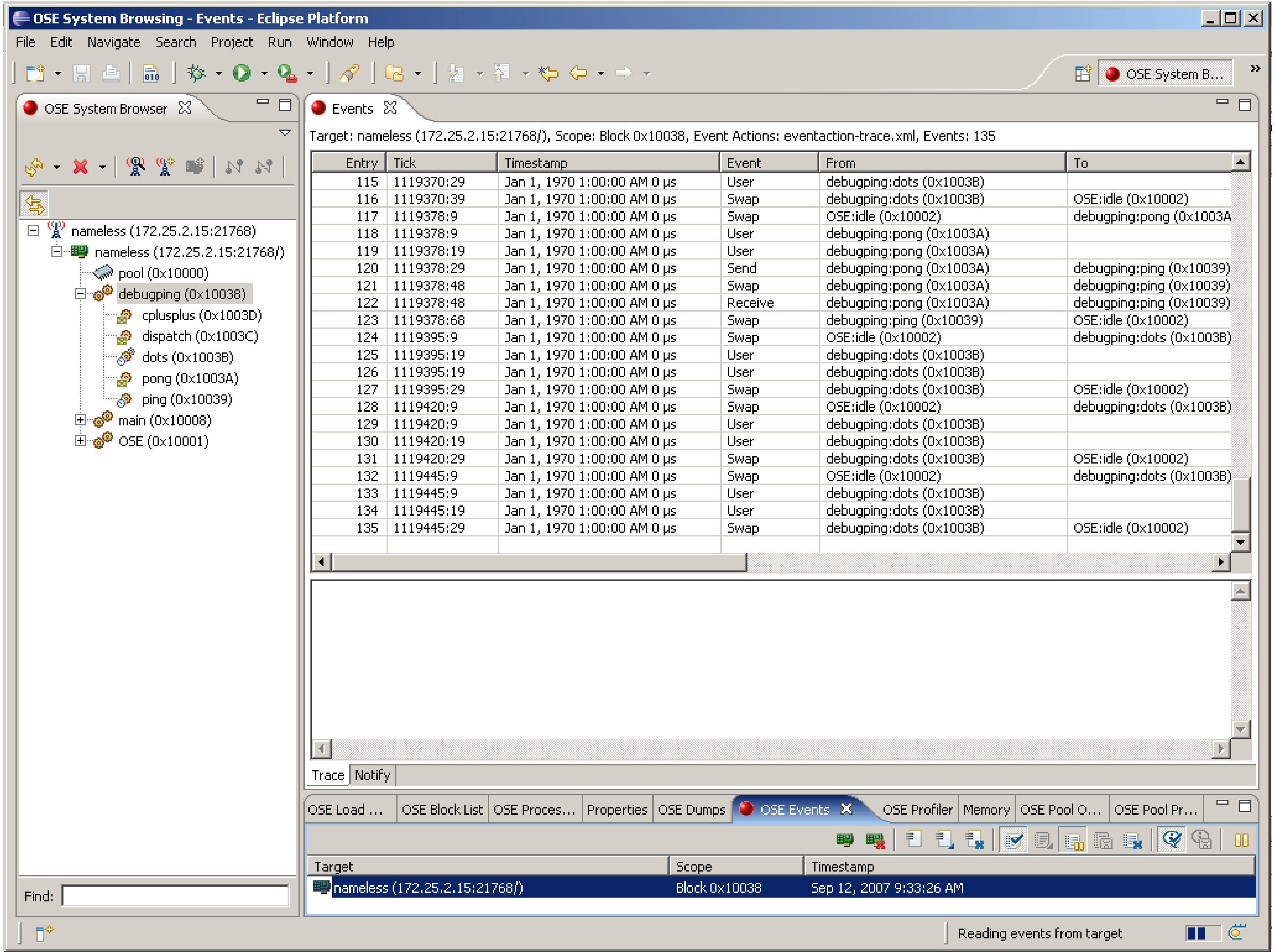
- Used to implement tracing of CRT calls in OSE
    - Each CRT function has unique event numbers for function entry and exit
    - Functions with significant data has an event number for the payload
    - File system accesses can be traced from the application level, through signal transaction, and down to the device driver
    - For example heap calls malloc() and free() are instrumented with user events allowing you to trace heap misuse
- Any application warnings or logging needs you can think of!

Event Tracing

- Log OSE system events or application events
- Filter which event to trace (events, processes involved, etc)
- All events have meta data about time and current process
- Event data such as signal data or free text from application
- Trace data can be saved in text files for further processing
- Trace data uploaded and displayed in Eclipse tools

Event Breakpoints

- Stop application on specified system or application events
- Select which applications/processes to stop
- Show data about the event in Eclipse tools



Target: nameless (172.25.2.15:21768/), Scope: Block 0x10038, Event Actions: eventaction-trace.xml, Events: 135

Entry	Tick	Timestamp	Event	From	To
115	1119370:29	Jan 1, 1970 1:00:00 AM 0 µs	User	debugping:dots (0x1003B)	
116	1119370:39	Jan 1, 1970 1:00:00 AM 0 µs	Swap	debugping:dots (0x1003B)	OSE:idle (0x10002)
117	1119378:9	Jan 1, 1970 1:00:00 AM 0 µs	Swap	OSE:idle (0x10002)	debugping:pong (0x1003A
118	1119378:9	Jan 1, 1970 1:00:00 AM 0 µs	User	debugping:pong (0x1003A)	
119	1119378:19	Jan 1, 1970 1:00:00 AM 0 µs	User	debugping:pong (0x1003A)	
120	1119378:29	Jan 1, 1970 1:00:00 AM 0 µs	Send	debugping:pong (0x1003A)	debugping:ping (0x10039)
121	1119378:48	Jan 1, 1970 1:00:00 AM 0 µs	Swap	debugping:pong (0x1003A)	debugping:ping (0x10039)
122	1119378:48	Jan 1, 1970 1:00:00 AM 0 µs	Receive	debugping:pong (0x1003A)	debugping:ping (0x10039)
123	1119378:68	Jan 1, 1970 1:00:00 AM 0 µs	Swap	debugping:ping (0x10039)	OSE:idle (0x10002)
124	1119395:9	Jan 1, 1970 1:00:00 AM 0 µs	Swap	OSE:idle (0x10002)	debugping:dots (0x1003B)
125	1119395:19	Jan 1, 1970 1:00:00 AM 0 µs	User	debugping:dots (0x1003B)	
126	1119395:19	Jan 1, 1970 1:00:00 AM 0 µs	User	debugping:dots (0x1003B)	
127	1119395:29	Jan 1, 1970 1:00:00 AM 0 µs	Swap	debugping:dots (0x1003B)	OSE:idle (0x10002)
128	1119420:9	Jan 1, 1970 1:00:00 AM 0 µs	Swap	OSE:idle (0x10002)	debugping:dots (0x1003B)
129	1119420:9	Jan 1, 1970 1:00:00 AM 0 µs	User	debugping:dots (0x1003B)	
130	1119420:19	Jan 1, 1970 1:00:00 AM 0 µs	User	debugping:dots (0x1003B)	
131	1119420:29	Jan 1, 1970 1:00:00 AM 0 µs	Swap	debugping:dots (0x1003B)	OSE:idle (0x10002)
132	1119445:9	Jan 1, 1970 1:00:00 AM 0 µs	Swap	OSE:idle (0x10002)	debugping:dots (0x1003B)
133	1119445:9	Jan 1, 1970 1:00:00 AM 0 µs	User	debugping:dots (0x1003B)	
134	1119445:19	Jan 1, 1970 1:00:00 AM 0 µs	User	debugping:dots (0x1003B)	
135	1119445:29	Jan 1, 1970 1:00:00 AM 0 µs	Swap	debugping:dots (0x1003B)	OSE:idle (0x10002)

Target	Scope	Timestamp
nameless (172.25.2.15:21768/)	Block 0x10038	Sep 12, 2007 9:33:26 AM

Reading events from target

Are you still awake?

Report Profiling – A flexible profiling system

Report Profiling

- Reports contains information about usage over time
- A report contains statistic for a configurable integration period
- Values represented as signed integers, suitable for percent and amount
- The data can be one or two dimensional
- Reports are generated periodically by the OS and stored in a circular buffer (of a dynamic configurable size)
- Clients (host or target based) read continually and receives chunks of reports in an efficient way
- Open and documented client API for configuring and reading reports

Report Profiling

- Different types of Report Profiling
  - CPU usage per
    - CPU (per core in SMP systems)
    - Process priority (interrupt, 0-31, and background)
    - Process (thread) (configurable max number per report)
    - Program
  - Heap usage per:
    - Process (configurable max number)
      - top users
  - User defined (OS provided API)

CPU Report Profiling

- CPU Usage per core (in SMP systems)
- CPU Usage per priority level, including interrupt level
- CPU Usage per process
  - top users
  - specified ID
  - specified name
  - system processes shown as sum
- Resolution only limited by hardware clock
- Two measurement principles interrupt sampled or recording context switches
- Statistics can be saved in text files for further processing
- Statistics presented in graphs in Eclipse tools

User Report Profiling

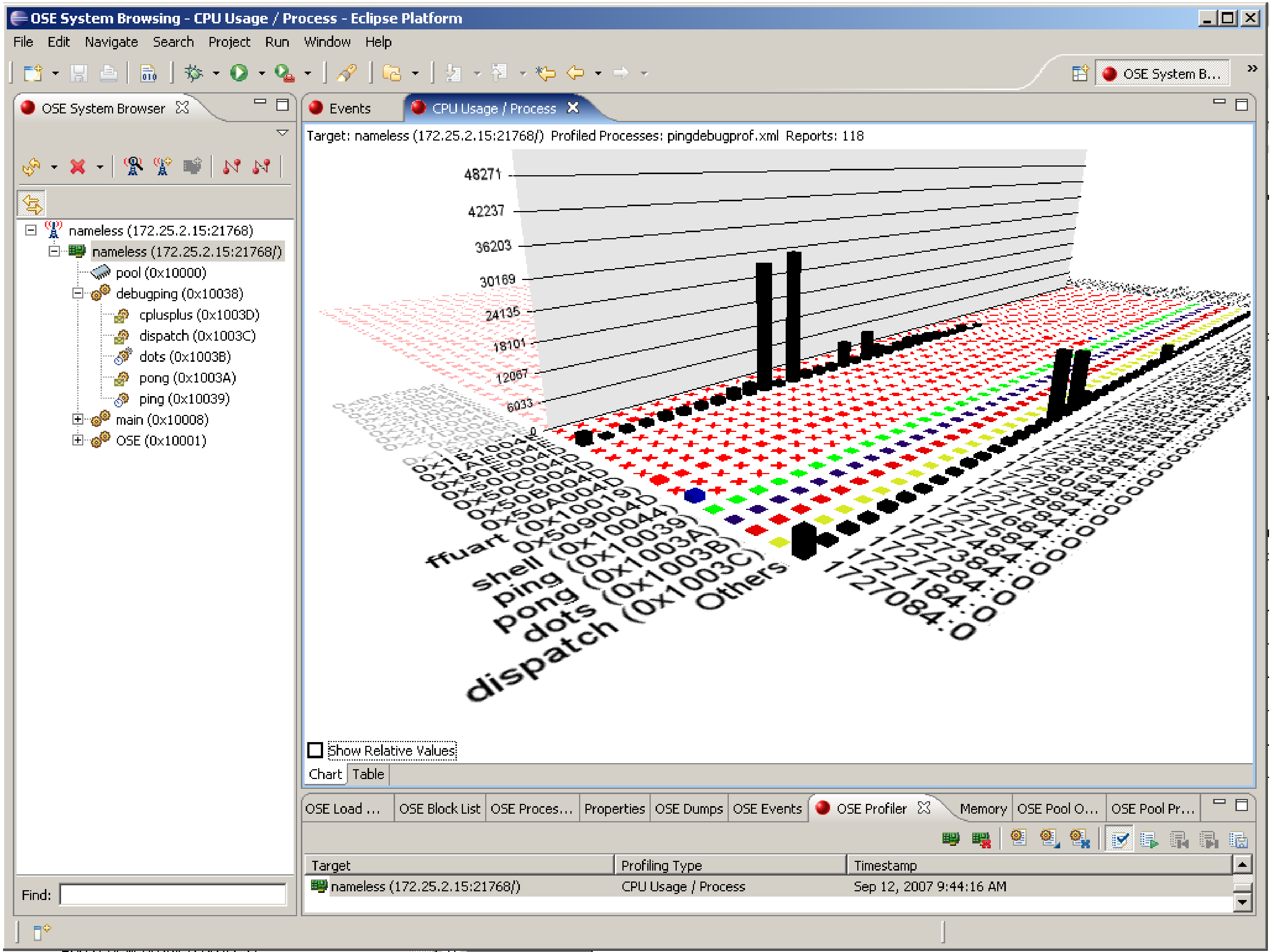
- Measuring type identifier is allocated by user
- Almost zero intrusiveness when not used
- Single value or value per object (two dimensional)
- Optionally the maximum and minimum value per interval can be collected
- Simple API provided by the OS, only two functions:

```
ose_create_report( SIGSELECT reportno, OSADDRESS *trig,
                   OSBOOLEAN multiple, OSBOOLEAN
                            continuous, OSBOOLEAN
maxmin);
ose_set_report_val(SIGSELECT reportno, OSREPORTID id,
                   OSREPORTVAL change);
```

User Report Use Cases

- Measure
  - Network I/O bandwidth utilization
  - Different types of memory consumption
  - File system utilization
  - Hardware registers
    - Collect statistics from hardware counters
    - Hardware automatically read when integration period ends
  - Any resource statistic or application numbers you can think of!

- Load balancing applications can use this for distributing jobs in a cluster
- Using Optima tools for pinpointing bottlenecks and optimization opportunities
- Visualize what is going on in my complex distributed system



- nameless (172.25.2.15:21768)
  - nameless (172.25.2.15:21768/)
    - pool (0x10000)
    - debugping (0x10038)
      - cplusplus (0x1003D)
      - dispatch (0x1003C)
      - dots (0x1003B)
      - pong (0x1003A)
      - ping (0x10039)
    - main (0x10008)
    - OSE (0x10001)

Target: nameless (172.25.2.15:21768/) Profiled Processes: pingdebugprof.xml Reports: 118

Target	Profiling Type	Timestamp
nameless (172.25.2.15:21768/)	CPU Usage / Process	Sep 12, 2007 9:44:16 AM

Taking the Event and Profiling System to Linux

- LINX – The Signal and Link handler concept for OSE, OSEck, and LINUX!
  - Now is the simple and powerful signal API available for Linux including the hunt, attach, send, and receive functionality
  - Tightly integrated in the Linux kernel
  - Open and available to all from Sourceforge
  - The right OS for the right task and they can all talk to each other
- Next step is to bring the advanced event action system and report profiling to Linux
  - Integrate the Event system with LINX and the Linux kernel
  - Provide the user report profiling API for Linux
- Linux developers will benefit from powerful system tools previously only available for OSE!

Questions?

sven.lundblad@enea.se