A Universal Infrastructure for the Run-time Monitoring of Parallel and Distributed Applications

Roland Wismüller, Jörg Trinitis, Thomas Ludwig

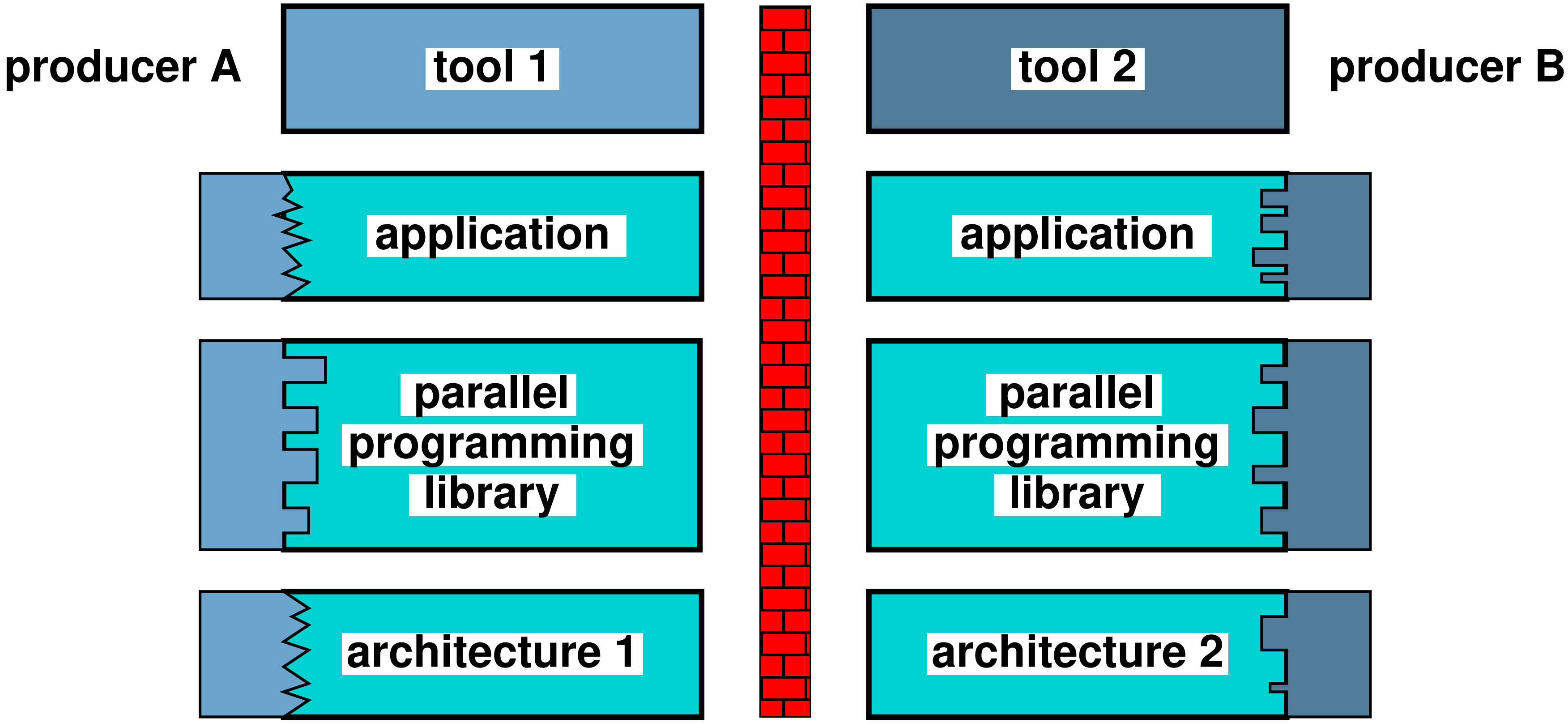
LRR-TUM
Institut für Informatik
Technische Universtität München
Germany

e-mail: wismuell@in.tum.de

Motivation

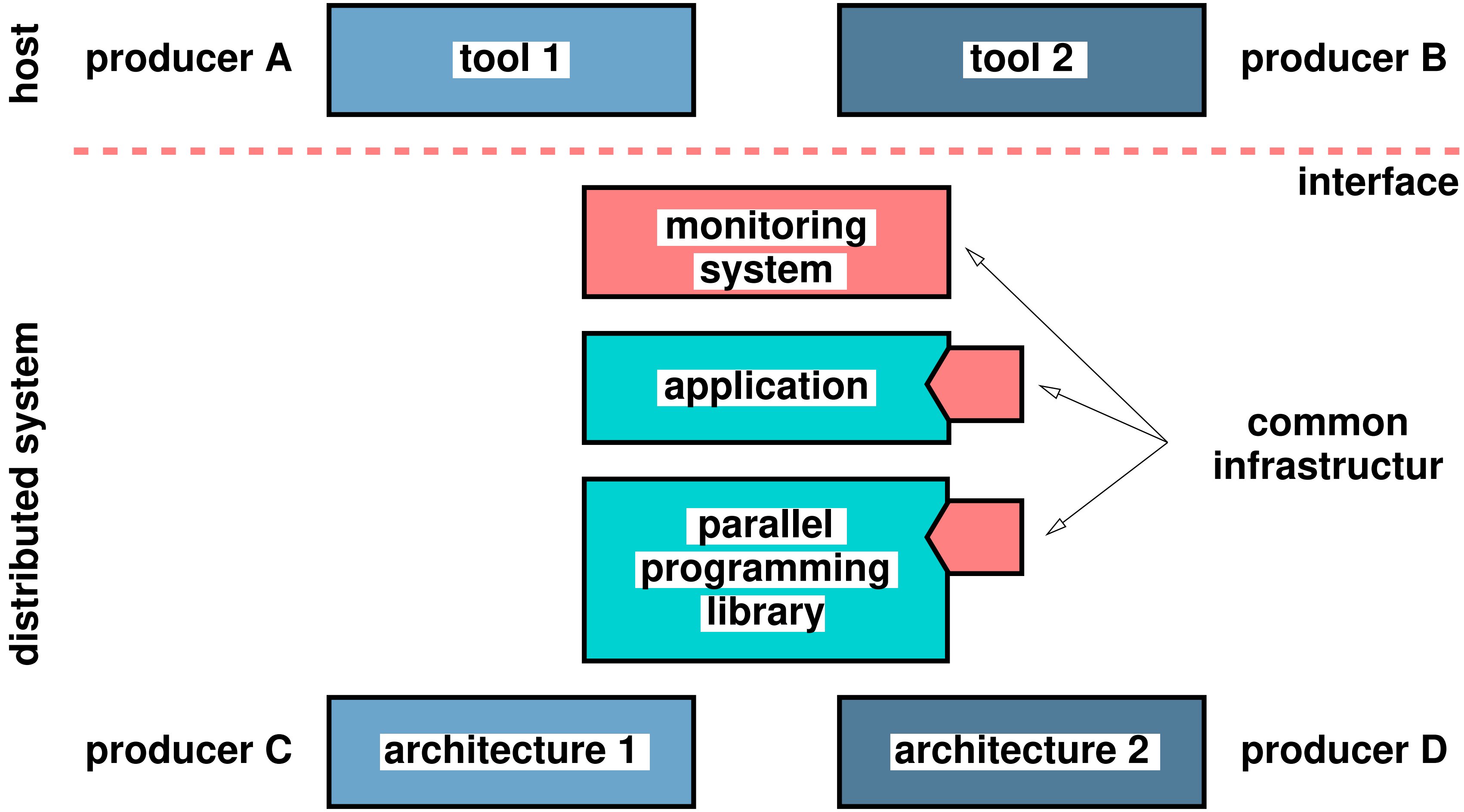
- On-line tools for parallel and distributed programs need an infrastructure for **observation** and **manipulation** of program executions
  - e.g. debugging, performance analysis, load balancing, resource management, adaptive run-time systems, computational steering ...
- On-line monitoring systems are difficult to build
  - existing tools use proprietary monitoring systems only usable by that tool
  - no standard infrastructure exists
- Consequences:
  - development of a new tool $\Rightarrow$ development of a new monitoring system
  - tools do not interoperate
  - tools are available only on a small number of platforms

Current Tool Design



- Different modifications / extensions make tools incompatible
- Modifications / extensions are system dependent

Tool Design using OMIS



- Common interface serves both tools (even at the same time)
- Reduced cost for porting (only one system must be ported)

OMIS: Requirements

The interface should ...

- support on-line monitoring
  - adjust at run-time what should be monitored
  - modify target's run-time behavior if desired
- be platform independent
  - platform means hardware, operating system, programming library
  - supports development of uniform tool environments
- be usable for any kind of run-time tool
  - observing and/or manipulating tools
  - interactive / automatic, centralized / distributed

OMIS: Requirements

The interface should ...

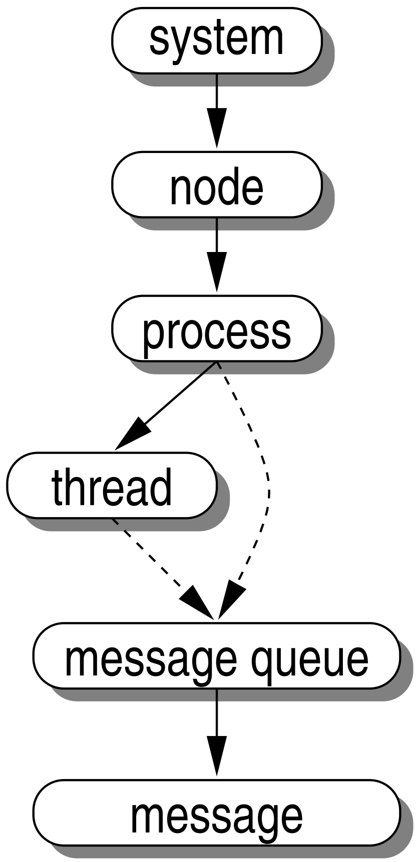
- support interoperable tools
  - sets of independent, small tools that can be used together
  - mechanisms for coordination between tools
- be efficient
  - minimal intrusion (CPU usage, communication)
  - scalability

☞ Meeting all of these requirements at the same time is hard

OMIS: Basic Concepts

Object based approach

- observed system is a hierarchical set of objects
  - abstract object classes: nodes, processes, threads, messages, and message queues
  - node and process model suitable for DMPs, NOWs, SMPs, and clusters of SMPs
- access via abstract identifiers (tokens)
- OMIS services observe and manipulate objects
  - OMIS core only defines platform independent services
  - other services are provided by platform specific extensions
- tools define their own view of the observed system



OMIS: Basic Concepts

Event / action paradigm

- monitoring requests consist of two parts
  - event definition specifies when something has to be done
  - action list specifies what has to be done
  - event definition may be empty $\Rightarrow$ execute action list immediately
  - tools differ mainly in the required event/action relations
  - events needed by different tools are fairly the same
  - actions may be more diverse $\Rightarrow$ extensions
- actions are autonomously executed by the monitoring system
  - monitoring system can be ‘programmed’ according to a tool’s needs
- event context parameters allow to pass information on the event occurrence (e.g. node, process, time) to the actions

OMIS: Basic Concepts

Scalability

- nearly all services operate on sets of objects
  - usage is more comfortable
  - allows more efficient communication and execution
- token expansion
  - object tokens can be expanded to the set of all contained objects
  - e.g. a node token can be used to represent all processes on this node

Location transparency

- tools do not need to care about the location of objects
- services operating on object sets do not make much sense otherwise
- token localization:
  - object token can be replaced by the token of the containing object
  - e.g. a process token can be used to represent the node it is located on

OMIS: Basic Concepts

Extensibility

- three reasons:
  - tools may require special actions and/or events
  - functionality of the tool should be executed in a distributed way, closer to the target objects
  - specific target platforms may require additional services and/or objects
- two kinds of extensions:
  - distributed tool extensions
    - independent of target system and OMIS implementation
    - new services exclusively based on existing ones
    - OMIS defines both external and internal interface
  - monitor extensions
    - may depend on target system and OMIS implementation
    - new kinds of services and/or objects
    - OMIS only defines the external interface

OMIS Request Examples

```
thread_reached_addr([p_14,p_52], 0x1234):
        thread_stop([]) thread_get_backtrace($thread, 0)
```

- When (a thread in) one of the specified processes reaches the breakpoint, stop the whole application and return the stack backtrace

```
thread_has_started_lib_call([p_21], "pvm_send"):
        counter_add(counter_1, $par5)
```

- When (a thread in) process p_21 starts a send call, add the message size (event context paramerter par5) to a counter

OCM – An Implementation of OMIS

OCM = OMIS compliant monitoring system for PVM on NOWs

Goals

- verify and evaluate the concepts of OMIS
- provide an infrastructure for tool developments
- provide a reference implementation of OMIS

Current state

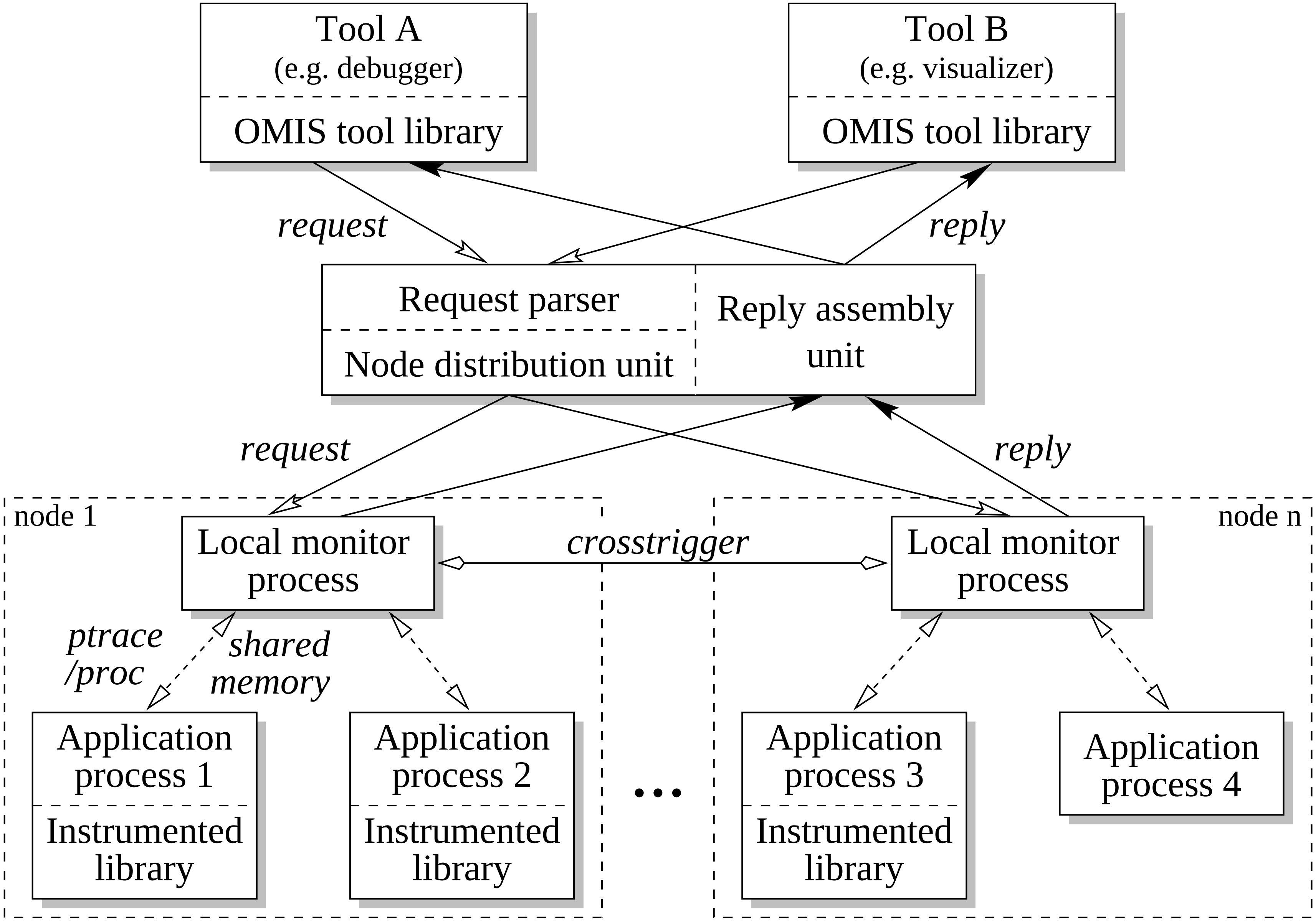
- first version is operational for PVM on Solaris, Linux, Irix
- first tool is available (debugger DETOP)
- ports to other platforms have begun (HPUX, AIX, MPI, TreadMarks)

OCM – An Implementation of OMIS

Design considerations

- distributed target system ⇒ distributed monitoring system
  - one monitor process per node, started on-demand
  - ☞ processes are independent servers with an OMIS-like interface
- global view provided by OMIS ⇒ additional (logically) central component
  - analyze requests, split it into pieces that can be processed locally
  - assemble local results
  - ☞ currently implemented as a central process (NDU)
- distributed monitoring system ⇒ communication
  - monitor processes must not block in receive calls
  - polling consumes too much CPU time
  - ☞ we need handler-driven receives (interrupts / Active Messages)

Global Structure of OCM



Request Distribution in the OCM

Problem

- services can operate on sets of objects
- sets may be arbitrarily distributed
- request may contain services operating on different sets
  - may require communication for conditional requests

Solution

- NDU analyzes requests at definition time
- try to avoid communication where possible
- insert communication actions and events where necessary

Constraint

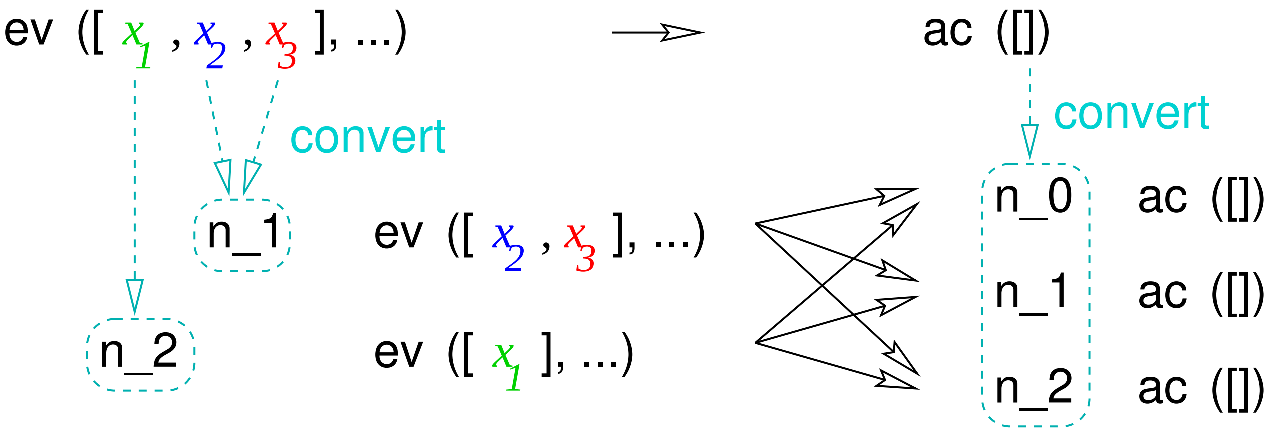
- NDU does not know the semantics of objects and services

Request Distribution: Example

OMIS request

- `thread_reached_addr([p_1, p_2, p_3], 0x234): thread_stop([])`
- for the NDU: $\mathrm{ev}([x_1, x_2, x_3], \ldots) \to \mathrm{ac}([])$

Distribution process

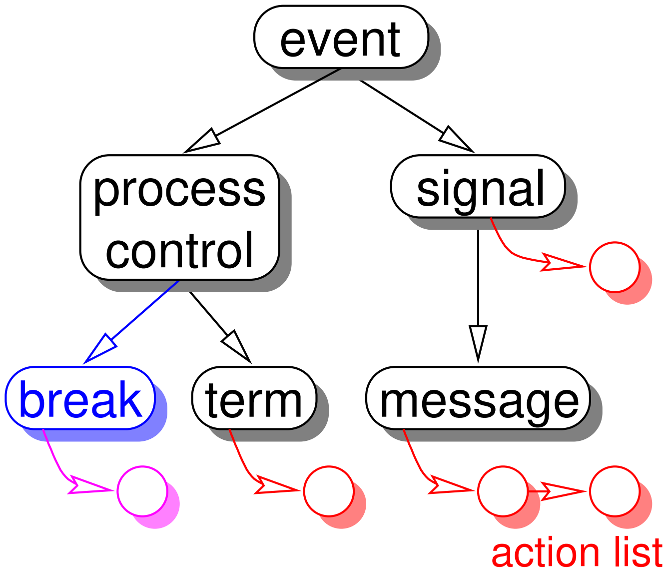


n_0	ct_recv(...) $\longrightarrow$ ac ([])	
n_1	ct_recv(...) $\longrightarrow$ ac ([])	ev ([x_2 , x_3], ...) $\longrightarrow$ ct_send ([], ...)
n_2	ct_recv(...) $\longrightarrow$ ac ([])	ev ([x_1], ...) $\longrightarrow$ ct_send ([], ...)

Local Monitor Process

Event driven operation

- wait for an event
- analyze event using a hierarchy of filters
- execute actions stored for this event



Special action for processing a request

- unconditional request
  - execute it and send back results
- conditional request
  - install new filter if necessary, insert or activate instrumentation
  - append action list to proper filter

Local Monitor Process

Efficiency problem

- requests like

```
thread_has_started_lib_call([p_1],"..."): timer_start(timer_1)
thread_has_ended_lib_call([p_1],"..."): timer_stop(timer_1)
```

  should be executed without context switching

- event detection and action execution must also be possible in the application process

Solution

- instrumented libraries contain parts of the monitoring system
- filter information and action lists are stored in a shared memory segment
- requests are analyzed whether they can be executed in the application process

Conclusions

Experience

- OMIS is very flexible and easy to use, but implementation is not trivial
  - optimal request distribution (protocols, consistency)
  - object sets can change dynamically
  - synchronization of shared memory access
  - extensibility (no assumptions on semantics $\Rightarrow$ need general solutions)

Future work

- finish implementation of OCM (NDU + a few services)
- provide OCM distribution (under GNU public license)
- interoperable tools and supporting infrastructure

Further information

http://wwwbode.informatik.tu-muenchen.de/~omis