

OCM — A Monitoring System for PVM

Thomas Ludwig

Roland Wismüller, Michael Oberhuber

Technische Universität München

Lehrstuhl Rechnertechnik und Rechnerorganisation (LRR-TUM)

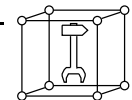
Institut für Informatik, 80290 München

Tel.: +49-89-289-22042

<http://wwwbode.informatik.tu-muenchen.de/~ludwig>

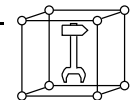
<http://wwwbode.informatik.tu-muenchen.de/~omis>

email: ludwig@informatik.tu-muenchen.de



Agenda

- Motivation
- OMIS — On-line Monitoring Interface Specification
- The Monitoring Interface
- Example: Performance Analysis
- Design Concepts
- The Software Module Structure
- Request Processing
- The Node Status Reporter (NSR)
- Current Status
- Future Work



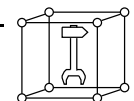
Motivation

Powerful tools for parallel and distributed programming need on-line monitoring facilities for **observation** and **manipulation** of programs during runtime.

(debugging, performance analysis, load balancing, computational steering...)

Sophisticated tools already exist, **but**:
all use proprietary monitoring systems which are incompatible to each other!

No standard infrastructure exists to connect tools to running systems
⇒ every new tool requires to implement a new monitoring system



OMIS — On-line Monitoring Interface Specification

Separate tool development from monitoring system development

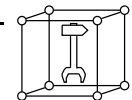
- ⇒ R&D groups involved in tool design and implementation
- ⇒ R&D groups involved in monitor design and implementation

Eventually have interoperable tools

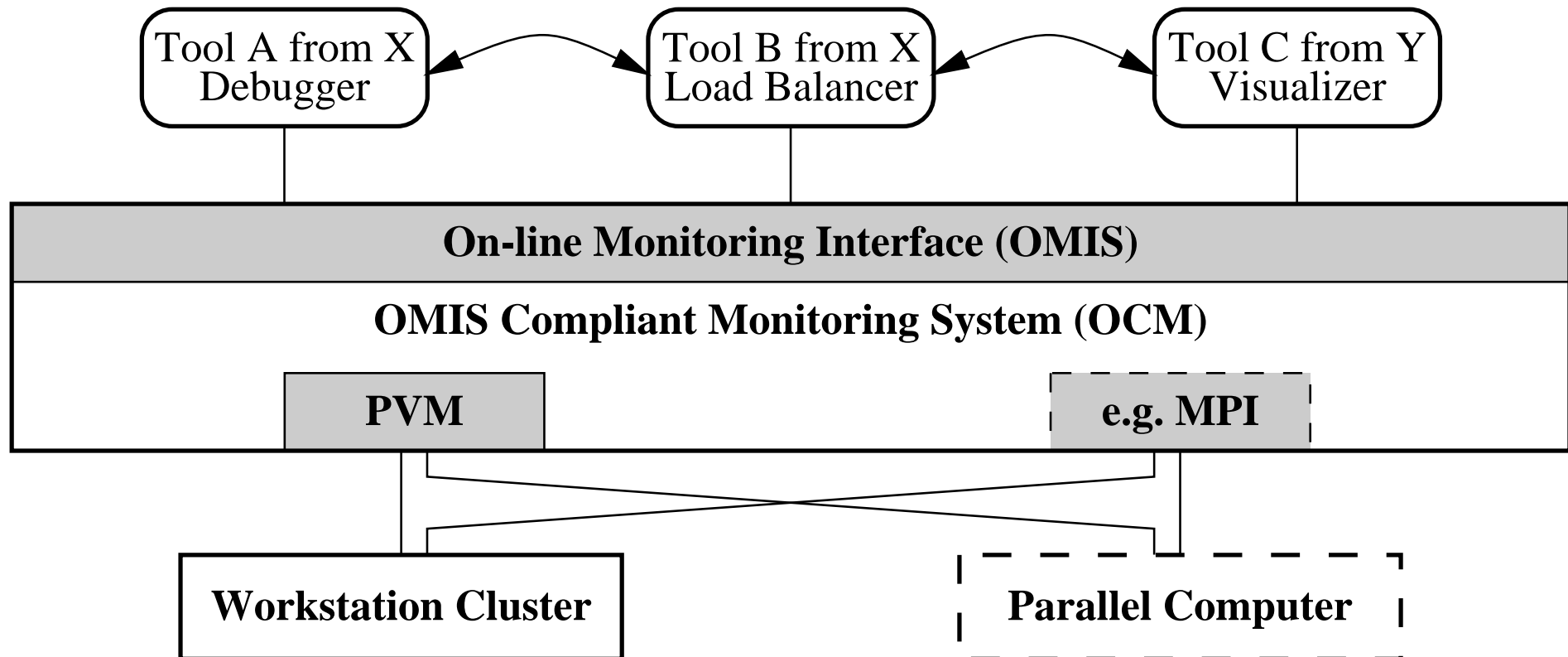
- ⇒ based on common monitoring layer
- ⇒ switch e.g. from visualizer to debugger

Eventually have uniform tools

- same set of tools on all target architectures for which an OMIS compliant monitoring system is implemented



OMIS Based Tool Environments

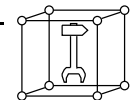


The Monitoring Interface (I)

OMIS offers services that are split into requests and (any number of) replies:

- Manipulation services
- Synchronous services
- Asynchronous services (events)

General syntax of requests (based on event-action scheme):

$$\begin{array}{lll} \textit{request} & ::= & [\textit{event} :] \textit{action_list} \\ \textit{action_list} & ::= & \textit{action1}, \textit{action2}, \dots \quad (\text{parallel execution}) \\ & | & \textit{action1}; \textit{action2}; \dots \quad (\text{sequential execution}) \\ \textit{event/action} & ::= & \textit{comm_id} \textit{target_ids} \textit{service_name} (\textit{params}) \end{array}$$


The Monitoring Interface (II)

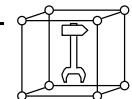
Two interface functions from tools to the monitoring system

Blocking

```
char* OMIS_request_block (char* request)
```

Non-blocking

```
void OMIS_request (char * request,  
                  void (* callback) (char* reply, void* param),  
                  void* callback_param);
```



Example: Performance Analysis

```
start_send(4178):  start_integrator(1),  add_counter(2,$3)
```

if process 4178 starts a sending call then start integrating counter 1 and add the third result parameter of the event occurrence (e.g. the number of bytes to be sent) to counter 2

```
end_send(4178):  stop_integrator(1)
```

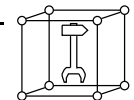
if process 4178 ends the sending call then stop the integrating counter 1

Result: integrating counter 1 keeps the time spend in sending calls, counter 2 keeps the number of bytes sent in total

Design Concepts

OMIS related design concepts

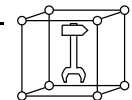
- Independence from the target platform:
Define generic monitoring system
All internal components use specified interfaces
- The monitoring system is a set of communicating monitors, one for each node of the virtual machine
- Event/action working model of (single node) monitor



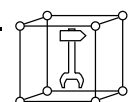
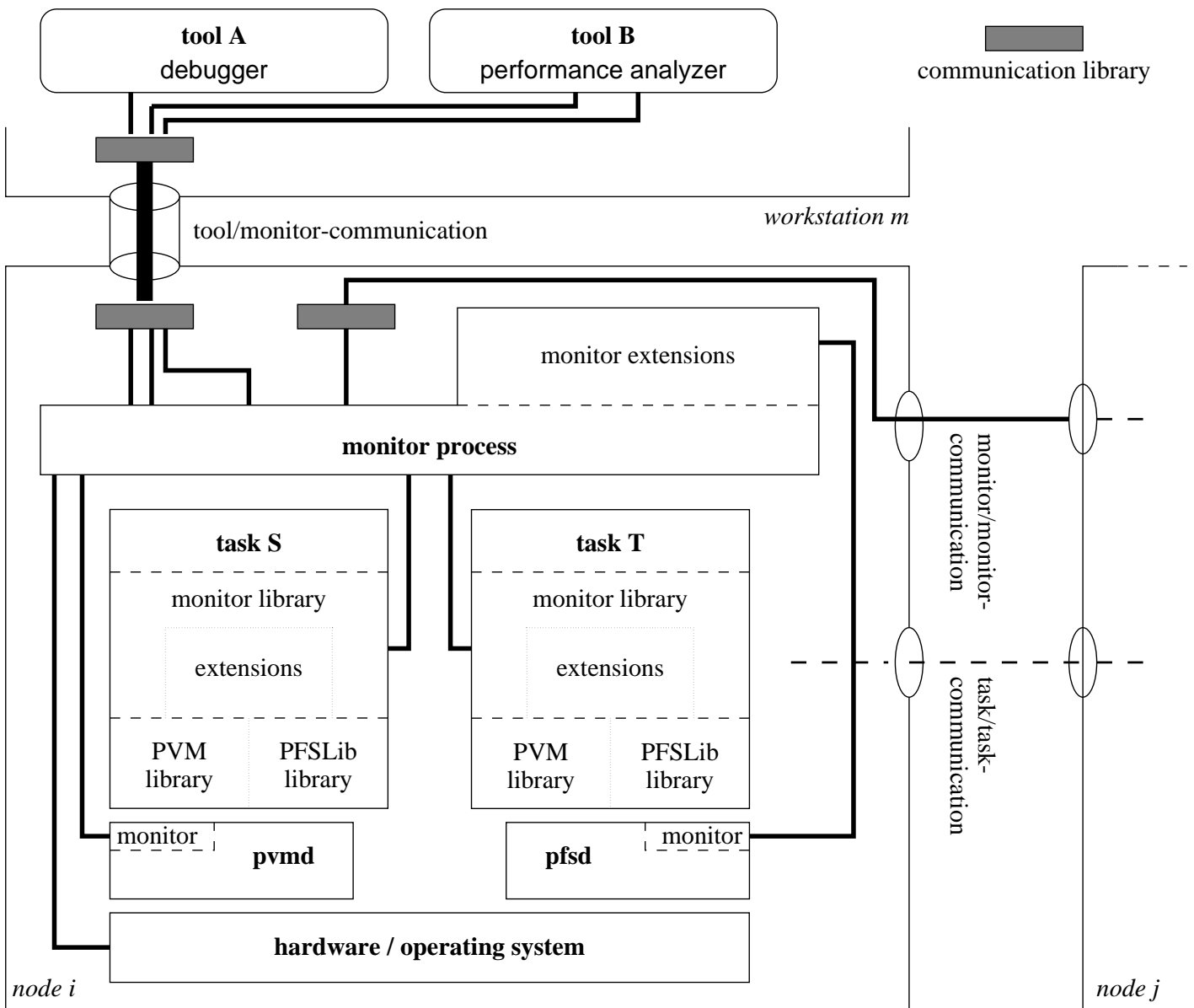
Design Concepts...

Implementation related design concepts

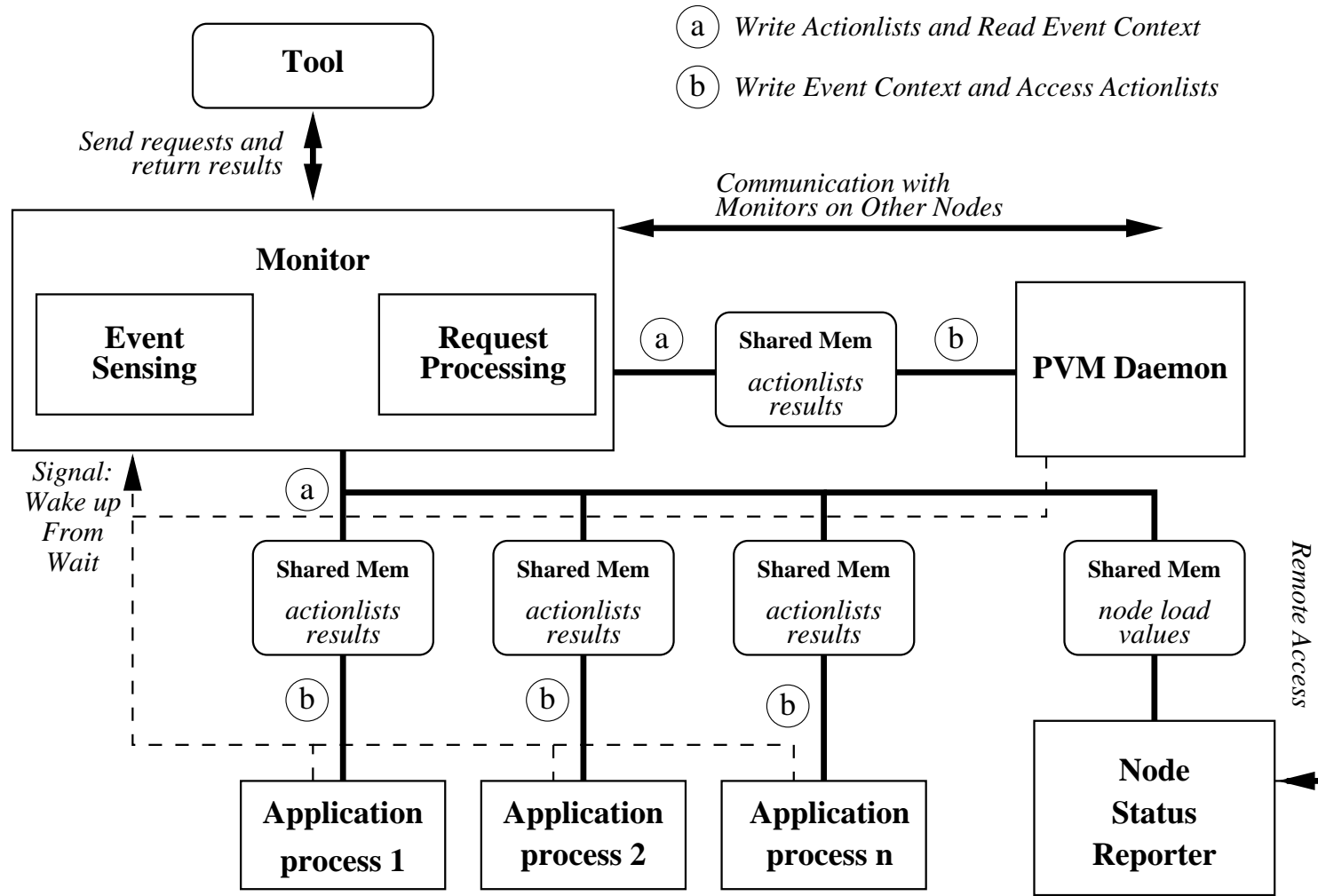
- Single node monitors is implemented as one single process
- Separate process for node status reporting; beneficial for load balancing and resource management
- Monitoring systems runs under user id; no special access rights necessary
- Global actions distributed to target nodes at definition time; execution started via automatically generated crosstrigger events
- Intermonitor-communication based on PVM
- Intramonitor-communication handled via shared memory segments



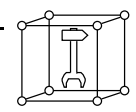
The Software Module Structure



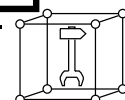
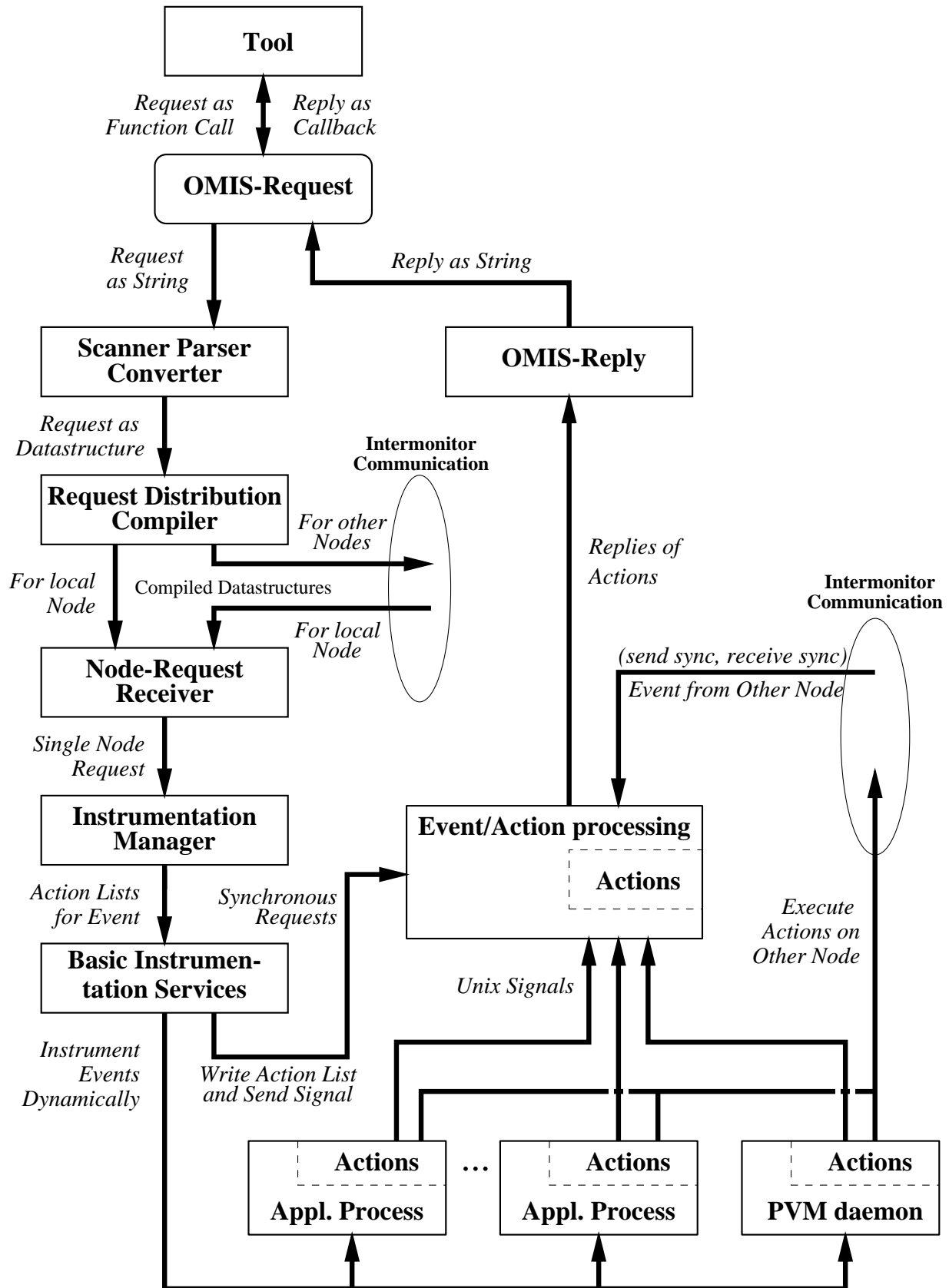
The Software Module Structure...



- Ⓐ Write Actionlists and Read Event Context
- Ⓑ Write Event Context and Access Actionlists



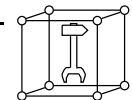
Request Processing



The Node Status Reporter (NSR)

Supports **load balancing** and **resource management**

- Responsible for collecting static and dynamic system information (available resources, load of resources)
- Send data on a regular basis via UDP broadcast and on individual request
- One NSR process per node installed by the system administrator
- Can be queried by any monitor



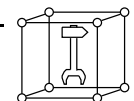
Current Status of OCM

3 researchers and 3 students working on the design since January 1996:

- Communication and Request Processing in OMIS Compliant Monitoring Systems (Michael Uemminghaus)
- Information and Event/Action Management in OMIS Compliant Monitoring Systems (Hans-Günter Zeller)
- Development of the Program/Monitor Interface for an OMIS Compliant Monitoring System (Manfred Geischeder)
- Design of the Node Status Reporter (Matthias Hilbig)

Individual design documents due: October, 15

Publicly available OCM design document: mid November



Future Work

- Discussion of relevant research topics with other groups active in that field
- Improve OMIS document: include feedback from other researchers and changes that are caused by the OCM design
- Implement OCM according to the specification

ANY FEEDBACK ON OMIS AND OCM IS WELCOME!

