OMIS — On-line Monitoring Interface Specification

Roland Wismüller

LRR-TUM
Institut für Informatik
Technische Universtität München

e-mail: wismuell@informatik.tu-muenchen.de

Laufzeit-Werkzeuge

Laufzeit-Werkzeuge: Werkzeuge, die den Ablauf eines Programms beobachten und ggf. auch manipulieren können

Beispiele:

- Debugger, Leistungsanalyse-Werkzeuge
- Visualisierungswerkzeuge, *computational steering*
- Lastausgleich, Ressourcenverwaltung

Aufgaben solcher Werkzeuge:

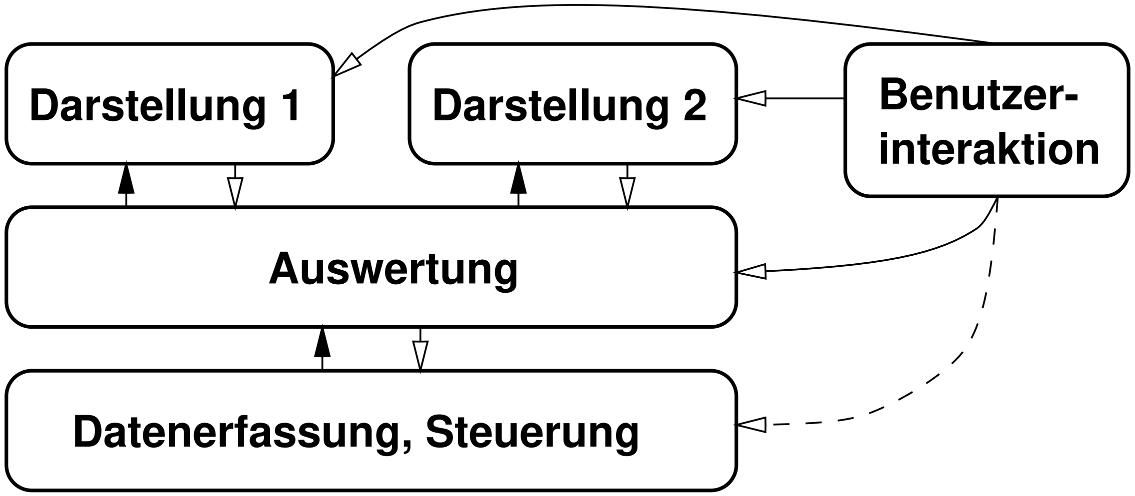
- Erfassung von Daten über das Ablaufverhalten eines Programms
- Auswertung und Darstellung dieser Daten

  ⤳ Wesentliche Unterschiede der Werkzeuge

- (Z.T.) Modifikation / Steuerung des Programmablaufs

Aufbau von Laufzeit-Werkzeugen

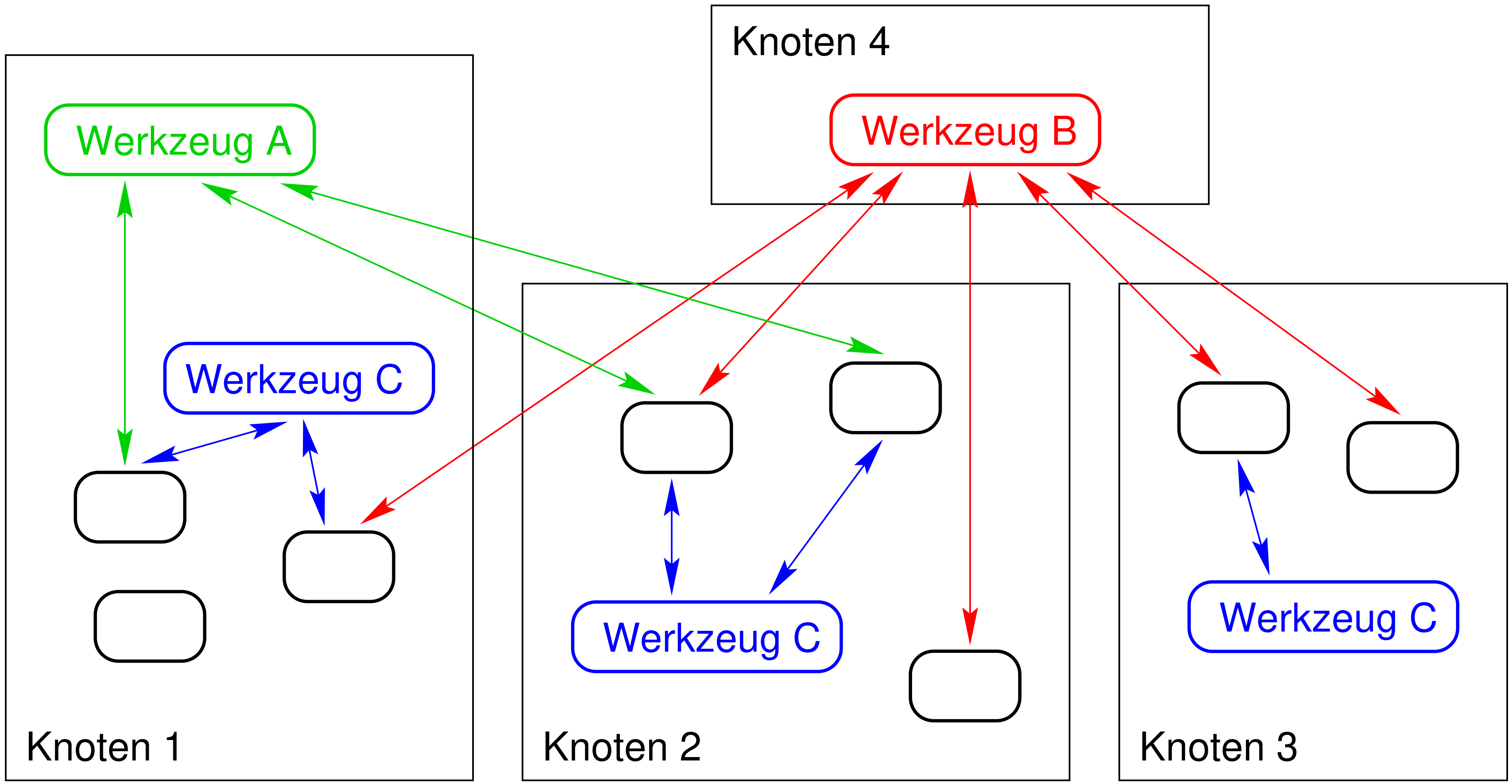
Struktur:



Einteilung der Werkzeuge in:

- *on-line* Werkzeuge: Auswertung und Darstellung erfolgen *zeitgleich* mit der Datenerfassung (und dem Ablauf des überwachten Programms)
  $\Rightarrow$ Einflußnahme auf Datenerfassung und Programmablauf möglich
- *off-line* Werkzeuge: Auswertung und Darstellung erfolgen *nach* dem Ablauf des überwachten Programms und der Datenerfassung.
  $\Rightarrow$ Daten werden in sog. *trace*-Dateien zwischengespeichert.

Ein typisches (Wunsch-)Szenario



Wünsche an Werkzeug-Umgebungen

Portabilität:

- Werkzeuge sollen einfach an verschiedene Zielsysteme anpaßbar sein.

$\leadsto$ aber: (effiziente) Datenerfassung ist oft sehr maschinenspezifisch!

Uniformität:

- identische Werkzeug-Umgebungen für alle Zielsysteme

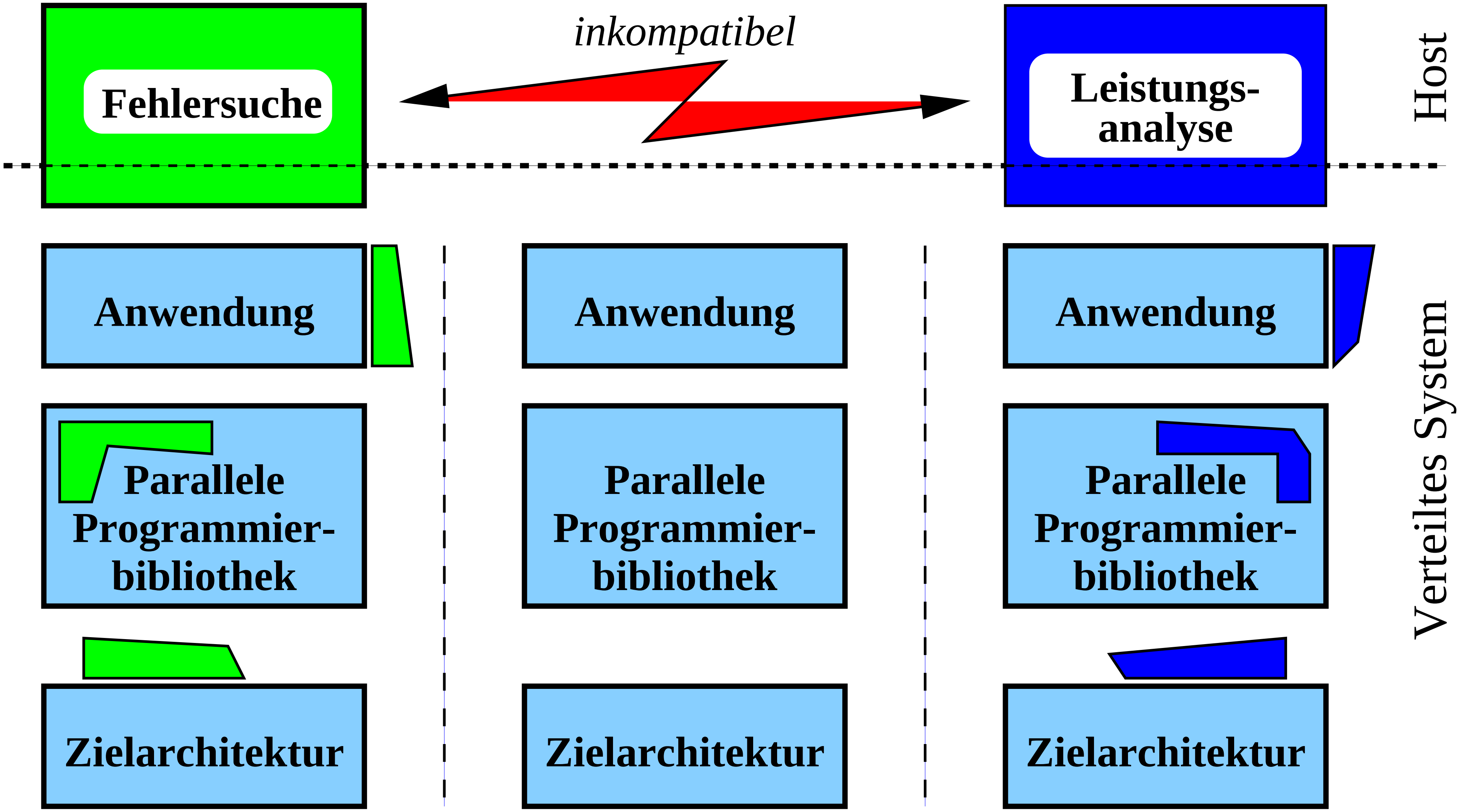
$\leadsto$ scheitert i.a. an der mangelnden Portierbarkeit

Interoperabilität:

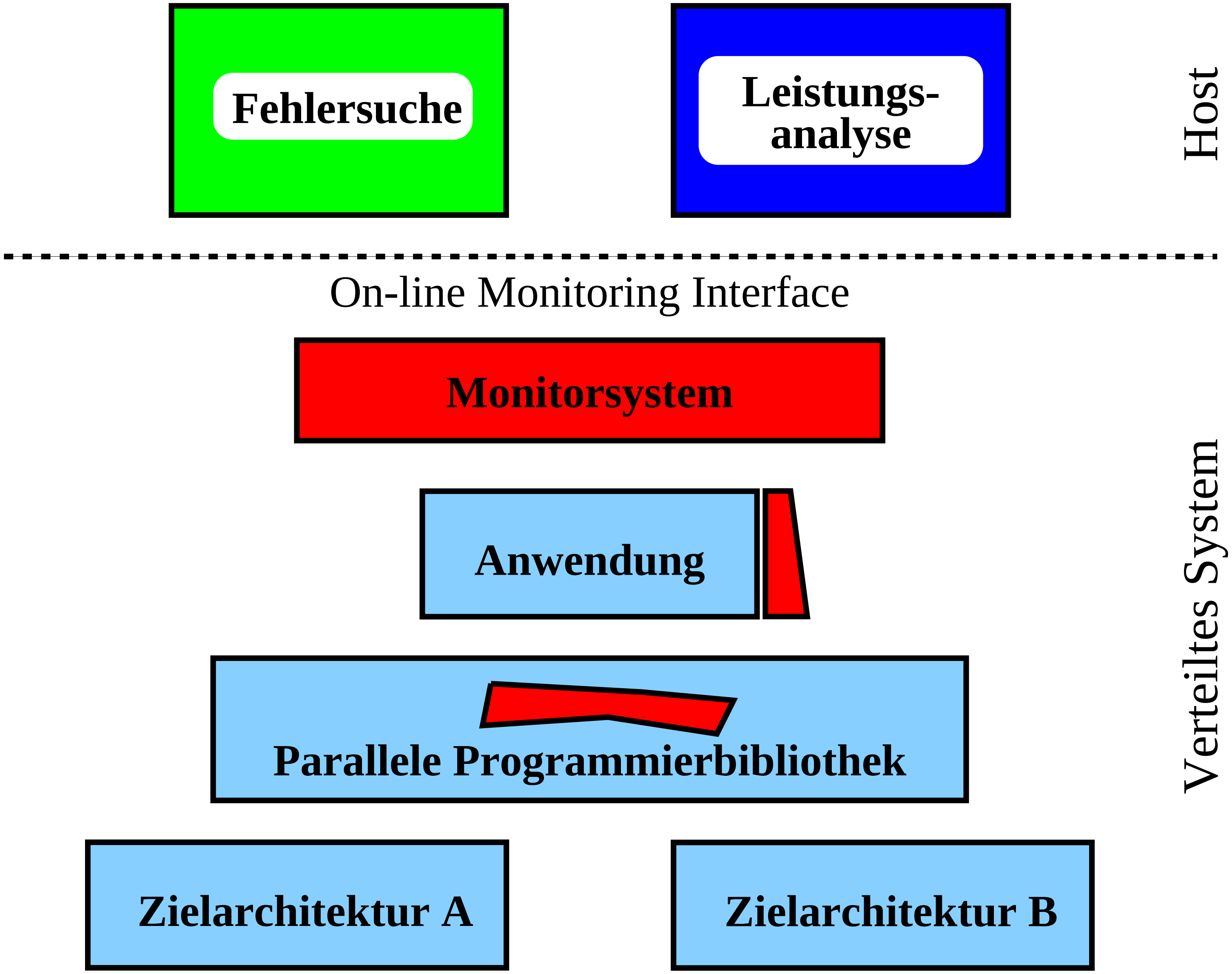
- verschiedene Werkzeuge sollten (gleichzeitig) auf dasselbe Programm anwendbar sein und sollten dabei kooperieren

$\leadsto$ scheitert i.a. an miteinander unverträglichen Datenerfassungstechniken

Konventionelle Werkzeugumgebungen



OMIS-basierte Werkzeugumgebungen



Erfüllung der Wünsche

Portabilität:

- Werkzeuge an sich portierbar durch Standard-Schnittstelle
- Monitorsystem weiterhin schwer portierbar, aber:
  - $\rightsquigarrow$ nur noch ein Monitorsystem pro Zielsystem erforderlich
  - $\rightsquigarrow$ Unabhängigkeit von Werkzeugentwicklung und Monitorentwicklung

Uniformität:

- Portierung des Monitorsystem $\Rightarrow$ Verfügbarkeit aller Werkzeuge
- neue Werkzeuge $\Rightarrow$ Verfügbarkeit auf allen OMIS-Plattformen

Interoperabilität:

- OMIS erlaubt Anschluß mehrerer Werkzeuge
- eingeschränkte Interaktion über OMIS Schnittstelle möglich

Anforderungen an die Schnittstelle

Allgemeingültigkeit und Flexibilität:

- verschiedene Werkzeuge zu verschiedenen Zwecken
- verschiedene Werkzeugarchitekturen
- verschiedene programmiersprachliche Objekte

Erweiterbarkeit:

- neue, zukünftige Werkzeugkonzepte
- neue zu überwachende Objekte
- neue Programmiermodelle

Effizienz

- wenig Kommunikation zw. Werkzeugen und Monitorsystem
- schnelle Datenerfassung und -verarbeitung

Struktur der Schnittstelle

Aus Sicht des Werkzeugs: (zentraler) Server, der Dienste anbietet

- drei Diensteklassen:
  - Informationsdienste (z.B. Liste der Prozesse)
  - Manipulationsdienste (z.B. halte Prozeß an)
  - Benachrichtigungsdienste (Ereignisse, z.B. Prozeß terminiert)
- Dienste sollen in einfacher Weise erweiterbar sein
  - z.B. werkzeugspezifische oder plattformspezifische Dienste

  $\leadsto$ benötigt genau festgelegte Erweiterungsschnittstelle (geplant, aber noch nicht in OMIS enthalten)
- Dienste arbeiten auf Objekten (genauer: Objektmengen), Strukturierung anhand der Objektklassen
  - in OMIS definiert: Knoten, Prozesse, Threads, Nachrichtenpuffer, Nachrichten, sowie Monitorobjekte

  $\leadsto$ auch Objektklassen müssen erweiterbar sein

Dienste in OMIS

1. **Knoten**
   - Überwachung einrichten / aufgeben, ...
   - statische Information (z.B. Hauptspeichergröße), dynamische Information (z.B. Last), ...

2. **Prozesse**
   - Erzeugung, Überwachung einrichten / aufgeben, Modifikation des Speichers, Ändern der Priorität, ...
   - statische Information (z.B. Argumente), dynamische Information (z.B. CPU-Zeit), Speicherinhalt, ...
   - Terminierung, Empfang eines Signals, ...

3. **Threads**
   - Überwachung aufgeben, Modifikation von Registern, Setzen des Programmzählers, Anhalten, Weiterführen, ...
   - statische Information (z.B. enthaltender Prozeß), dynamische Information (z.B. Scheduling-Zustand), Prozeduraufrufkeller, ...

Dienste in OMIS

3. **Threads (Forts.)**
   - Erzeugung von Prozessen / Threads, Hinzufügung / Löschung von Knoten zum Ausführungssystem, Stop, Weiterführung, Ausführung von Bibliotheksfunktionen, ...

4. **Nachrichtenpuffer und Nachrichten**
   - Nachricht aus Puffer löschen, Nachricht markieren, ...
   - Inhalt des Puffers, Sender einer Nachricht, ...
   - Eintrag einer Nachricht in einen Puffer, Empfang einer markierten Nachricht, ...

5. **Monitorobjekte**
   - Sperren / Freigeben / Löschen einer Dienstanforderung, ...
   - Information über vorhandene Dienste / Erweiterungen, ...

Dienstanforderungen in OMIS

Dienstanforderungen sind Ereignis-Aktions-Relationen:

- Ereignisdefinition: spezifiziert, *wann* etwas getan werden soll
- Aktionsliste: spezifiziert, *was* getan werden soll

Aktionsliste wird jedesmal ausgeführt, wenn das Ereignis eintritt

$\leadsto$ Parameterübergabemechanismus erlaubt Informationsfluß von Ereigniserkennung zu Aktionsausführung

$\leadsto$ Ereignisdefinition darf fehlen $\Rightarrow$ Aktionsliste wird sofort ausgeführt

Motivation:

- die Ereignisse, für die sich die Werkzeuge interessieren sind im Wesentlichen identisch, z.B. Prozeßerzeugung, Kommunikation
- die auszuführenden Aktionen können jedoch sehr werkzeugspezifisch sein, z.B. Protokollierung, Anhalten des Programms

Dienstanforderungen in OMIS

Dienstanforderungen sind Zeichenketten mit folgender Syntax:

request	::=	[*event_definition*] : *action_list*
action_list	::=	*action* | *action* [;] *action_list*
event_definition, action	::=	*service_name* (*parameters*)

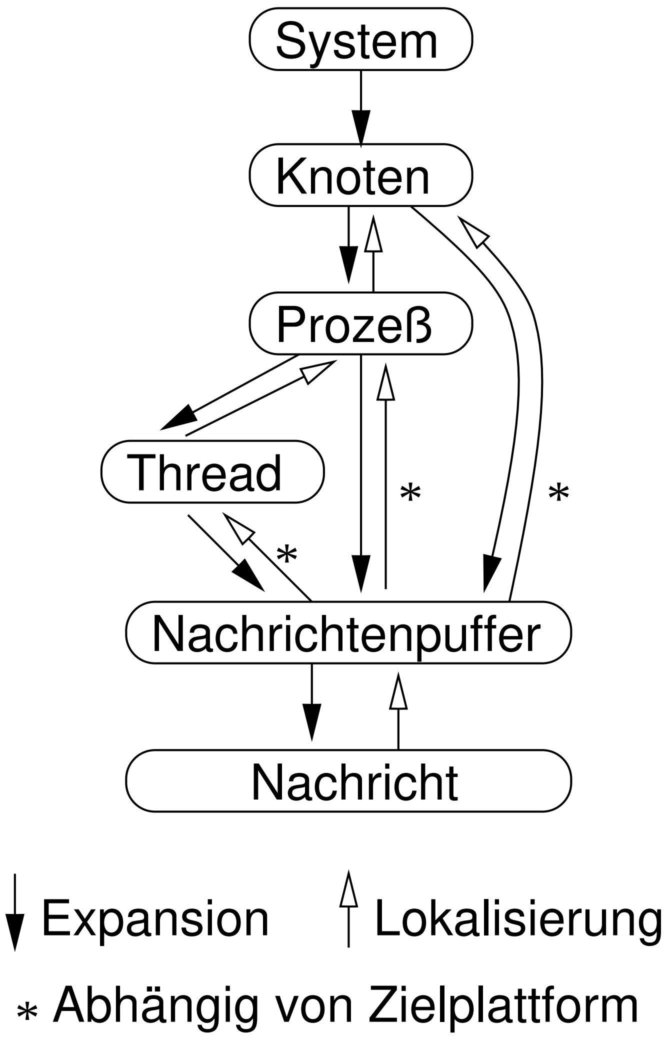
Semantische Details:

- (fast) alle Dienste arbeiten auf Objektmengen anstelle von Einzelobjekten ($\Rightarrow$ Skalierbarkeit)
- Objektbezeichnung erfolgt durch abstrakte Token ($\Rightarrow$ Systemunabhängigkeit)
- Ortstransparenz: Dienstanforderungen können sich auf beliebig verteilte Mengen von Objekten beziehen
- Aktionslisten können nebenläufig ausgeführt werden, eine Synchronisation (Barriere) ist durch ; möglich.

Tokenkonvertierung

- OMIS konvertiert übergebene Token automatisch in die vom Dienst erwarteten Klasse
- Objekthierarchie definiert zwei Konvertierungsrichtungen:
  - Expansion: z.B. Umwandlung eines Knotentokens in die Liste aller auf diesem Knoten vorhandenen Prozesse
  - Lokalisierung: z.B. Umwandlung eines Threadtokens in das Token des Knotens, auf dem der Thread liegt
- Jedes Werkzeug muß dazu seine eigene Sicht des überwachten Systems definieren

  ⇝ sonst z.B. Bedeutung von “alle Prozesse im System” unklar bzw. unsinnig
- Konvertierung erfolgt zur Ausführungszeit



Beispiel: Fehlersuche

```
thread_has_started_lib_call([],"pvm_send") :
    proc_get_info([$proc],12)  thread_get_backtrace([$thread],1)
```

Wenn irgendein Prozeß `pvm_send` aufruft: liefere seine `tid`, seinen Argumentenvektor und die Rücksprungadresse des Aufrufs

```
thread_reached_addr([p_12,p_34,p_44],0xfe08) : thread_stop([])
```

Wenn einer der spezifizierten Prozesse den Haltepunkt erreicht: halte die gesamte Anwendung an

Beispiel: Leistungsanalyse

```
thread_has_started_lib_call([p_21],"MPI_Send") :
    pt_timer_start(pt_t_1)  pt_counter_add(pt_c_1,$par5)
```

Wenn Prozeß p_21 einen Sendeaufruf startet: aktiviere einen Timer und addiere die Nachrichtenlänge ($par5) auf einen Zähler

```
thread_has_ended_lib_call([p_21],"MPI_Send") : pt_timer_stop(pt_t_1)
```

Wenn der Prozeß den Aufruf beendet: halte den Timer an

Resultat: Verweilzeit im Sendeaufruf und gesendete Nachrichtenmenge

Entstehungsgeschichte

- Erster Entwurf: Sommer 1995, LRR-TUM und V. Sunderam (Emory Univ.). Ziel: Monitorsystem für PVM auf NOWs
- OMIS 1.0 im WWW veröffentlicht im Feb. 1996
- Kommentare durch e-mail und persönliche Gespräche
- Konzept für Implementierung eines OMIS-konformen Monitorsystems
- Erarbeitung von OMIS 2.0 von Jan. 1997 bis Sept. 1997
  - Ortstransparenz, Token und Tokenkonversion
  - Änderungen in der Antwortstruktur
  - Vervollständigung der Spezifikation
- Lose, informelle Kooperationen mit anderen Universitäten:
  - Univ. Braunschweig, ENS Lyon, KFKI Forschungsinstitut Budapest, Univ. Linz, Rechenzentrum CYFRONET Krakau

Spezifikationsmethode

- Zweistufige Spezifikation:
  1. Syntax und Semantik von Dienstanforderungen
  2. Definition der einzelnen Dienste
- Keine formalen Beschreibungsmethoden für die Semantik; englische Prosa

  $\leadsto$ Referenzimplementierung als letztendliche Definition der Semantik
- Standardisierungsstrategie:
  - Veröffentlichung der Spezifikation im Internet
  - Bereitstellung der Referenzimplementierung
  - nur informelles Verfahren zur Aufnahme von Kommentaren und Änderungsvorschlägen
- Bisher keine Methodik zur Überprüfung der Konformität von Implementierungen

OCM – OMIS-konformes Monitorsystem

OCM = Implementierung von OMIS für das Programmiermodell PVM auf Netzen von Arbeitsplatzrechnern

Ziele:

- Realisierungsnachweis für OMIS
- Referenz für andere Implementierungen
- Implementierungsplattform für Werkzeuge Dritter

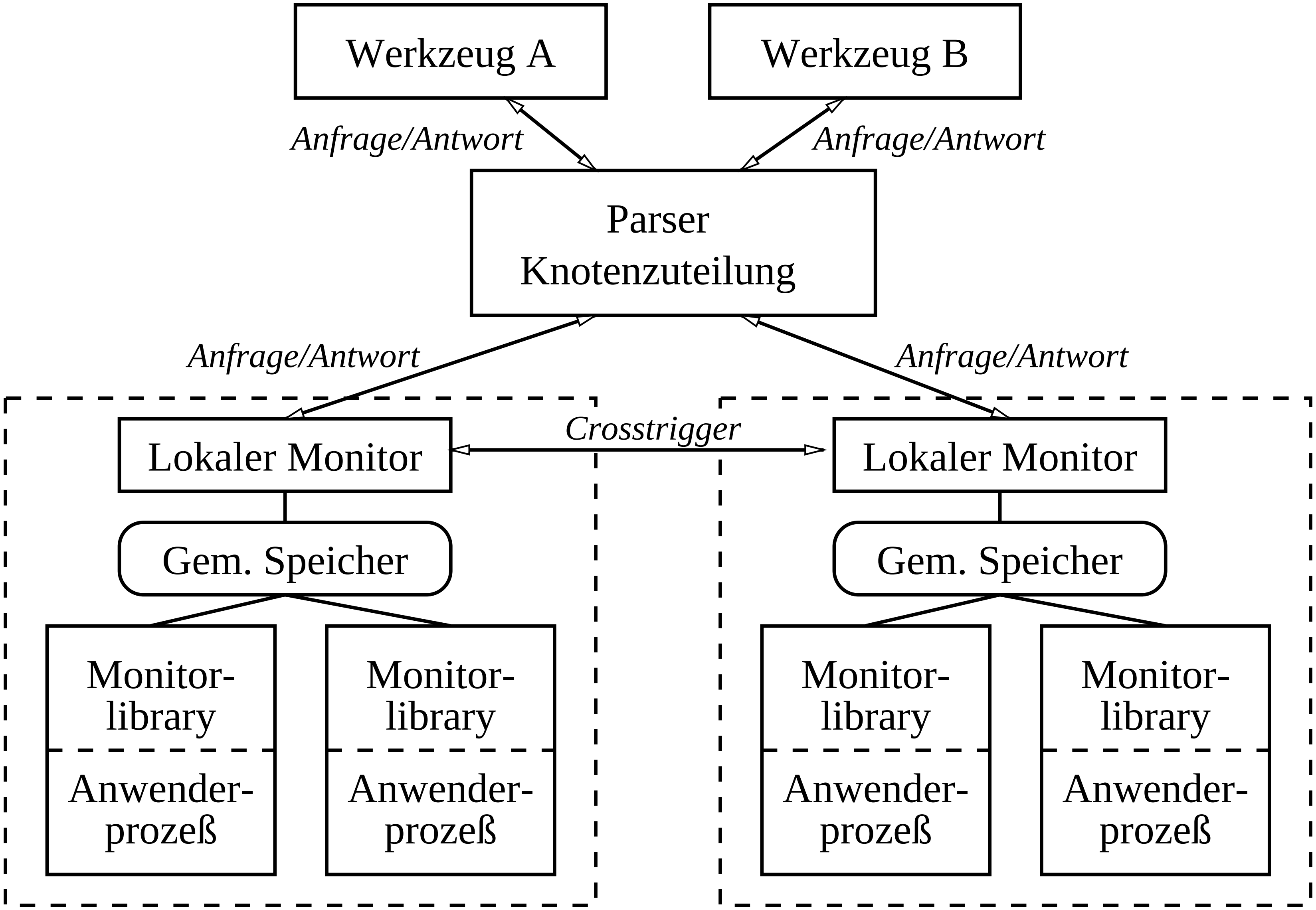
Status:

- Implementierung begonnen Jan. 1997, erster Prototyp Juli 1997
- noch nicht abgeschlossen
- wird unter GNU Lizenzbedingungen allgemein verfügbar gemacht

Entwurfskonzepte des OCM

- Monitorsystem besteht aus kommunizierenden Monitoren, jeweils einem pro Rechnerknoten des überwachten Systems
- Zusätzlich zentrale Verwaltungskomponente
  - Parsen der Dienstanforderungen
  - Verteilung globaler Dienstanforderungen an die Knoten zur Definitionszeit
    - $\leadsto$ bei der Ausführung von Aktionslisten werden nur noch Synchronisationsnachrichten (*crosstrigger*) versendet
- Ereignisgetriebenes Ausführungsmodell der (Einzelknoten-)Monitore
- Ereigniserkennung und Aktionsausführung sowohl im Monitorprozeß als auch in den Anwendungsprozessen
  - $\leadsto$ Kommunikation zwischen den knotenlokalen Monitorkomponenten über gemeinsamen Speicher und Signale

Grobstruktur des OCM



Weiterentwicklung

Monitorsystem:

- Abschluß der OCM-Implementierung, Anpassung von Werkzeugen
- Einsatz von OCM für andere Werkzeugumgebungen (Kooperationen)
- Anpassung von OCM an MPI

Schnittstellendefinition:

- Spezifikation der OMIS-Erweiterungsschnittstelle
- OMIS für Architekturen mit verteiltem gemeinsamem Speicher
- OMIS als (de-facto) Standard für verteiltes Monitoring