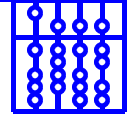


Konstruktion interoperabler Werkzeuge für parallele und verteilte Systeme

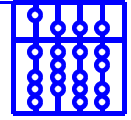
Thomas Ludwig

Lehrstuhl für Rechnertechnik und
Rechnerorganisation (LRR-TUM)
Institut für Informatik
Technische Universität München

ludwig@in.tum.de
www.in.tum.de/~ludwig



- ↳ **Werkzeugkonstruktion heute**
- ↳ **Methodische Werkzeugkonstruktion**
- ↳ **Schnittstellenbasierte Werkzeugkonstruktion**
- ↳ **Konzepte zur Interoperabilität**
- ↳ **Stand der Technik**
- ↳ **Erweiterung und Übertragung der Konzepte**
- ↳ **Status, Kooperationen, Anwendungen**
- ↳ **Zusammenfassung**

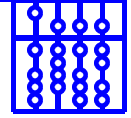


Anwendungsbereich

- ➔ Parallele Programme für Workstation-Cluster und Hochleistungsrechner

Der Software-Lebenszyklus

- ➔ Analyse — Spezifikation — Design —
Implementierung — Test — Produktion/Wartung



Besondere Problematik der parallelen Programme

- ➔ Nichtdeterminismus durch Kommunikationskonzepte
- ➔ Nichtreproduzierbarkeit durch zeitabhängiges Verhalten

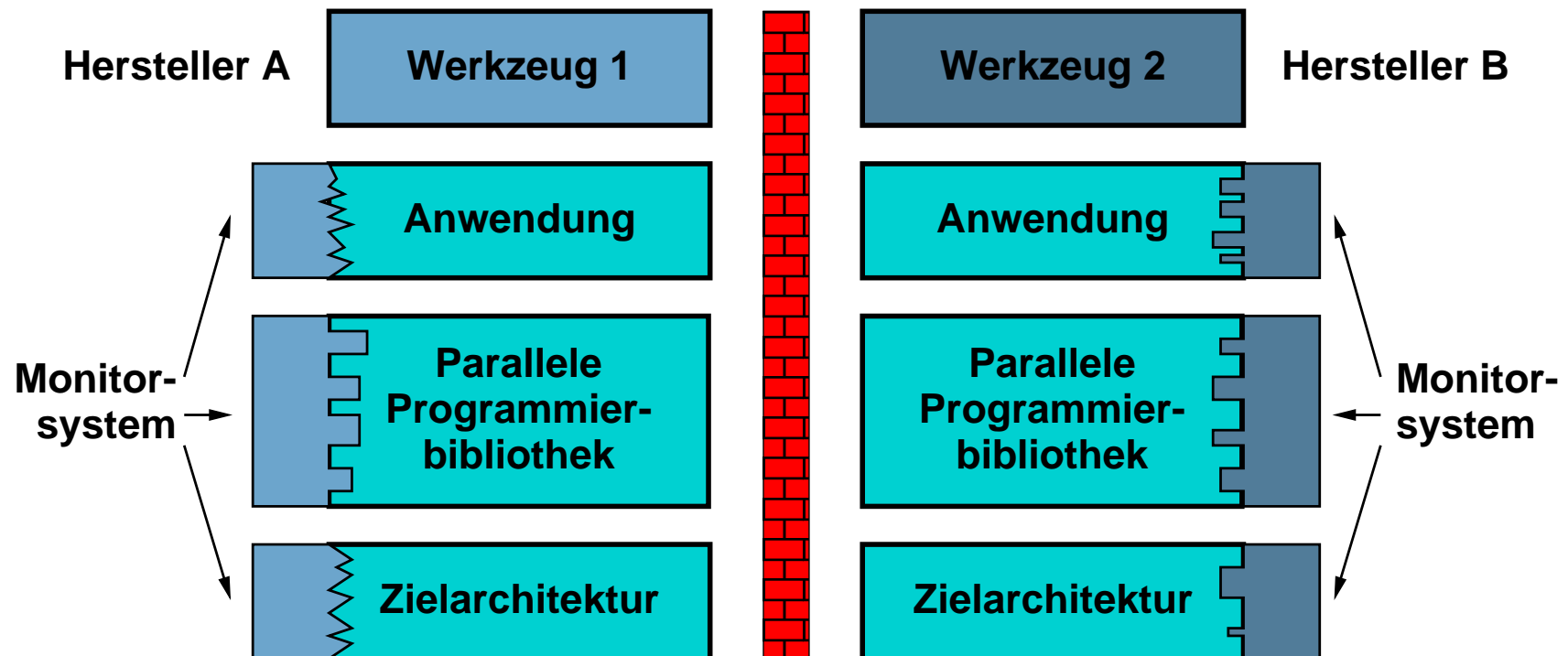
Notwendig: Online-Werkzeuge zur unmittelbaren Programmanipulation

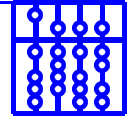
- ➔ Fehlersuche, Leistungsanalyse
- ➔ Deterministische Programmausführung, Berechnungssteuerung
- ➔ Sicherungspunktgeneratoren und automatische Lastverwaltung

Ausgangssituation ...

Werkzeugkonstruktion heute:

- ➡ Kein theoretisches Fundament (Betriebssysteme, verteilte Systeme)
- ➡ Ad-hoc-Entwurf mit monolithischem Aufbau



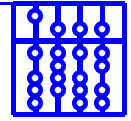


Aufwendige Werkzeugkonstruktion

- ➔ Ständige Neuentwicklung bereits vorhandener Funktionalitäten
- ➔ Benutzer hat schlechte Auswahl
- ➔ Konstruktion paralleler Software erschwert

Mangelhafte Konzepte bei den Werkzeugen

- ➔ Keine Kooperation möglich
Grund: Werkzeuge wissen nichts voneinander
- ➔ Keine gemeinsame Verwendung möglich
Grund: Werkzeuge verwenden verschieden modifizierte Systemkomponenten



Prinzip: Schnittstelle zwischen Werkzeug und Monitorsystem einziehen

Folge: Monitorsystem nur einmal realisieren; beliebige Werkzeuge aufsetzen

Die früheren Probleme sind damit eliminierbar

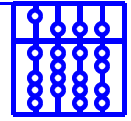
- ➔ Werkzeuge verwenden inkompatible Systemmodifikationen
- ➔ Werkzeuge wissen nichts voneinander

Gemeinsame Schnittstelle für alle Werkzeuge

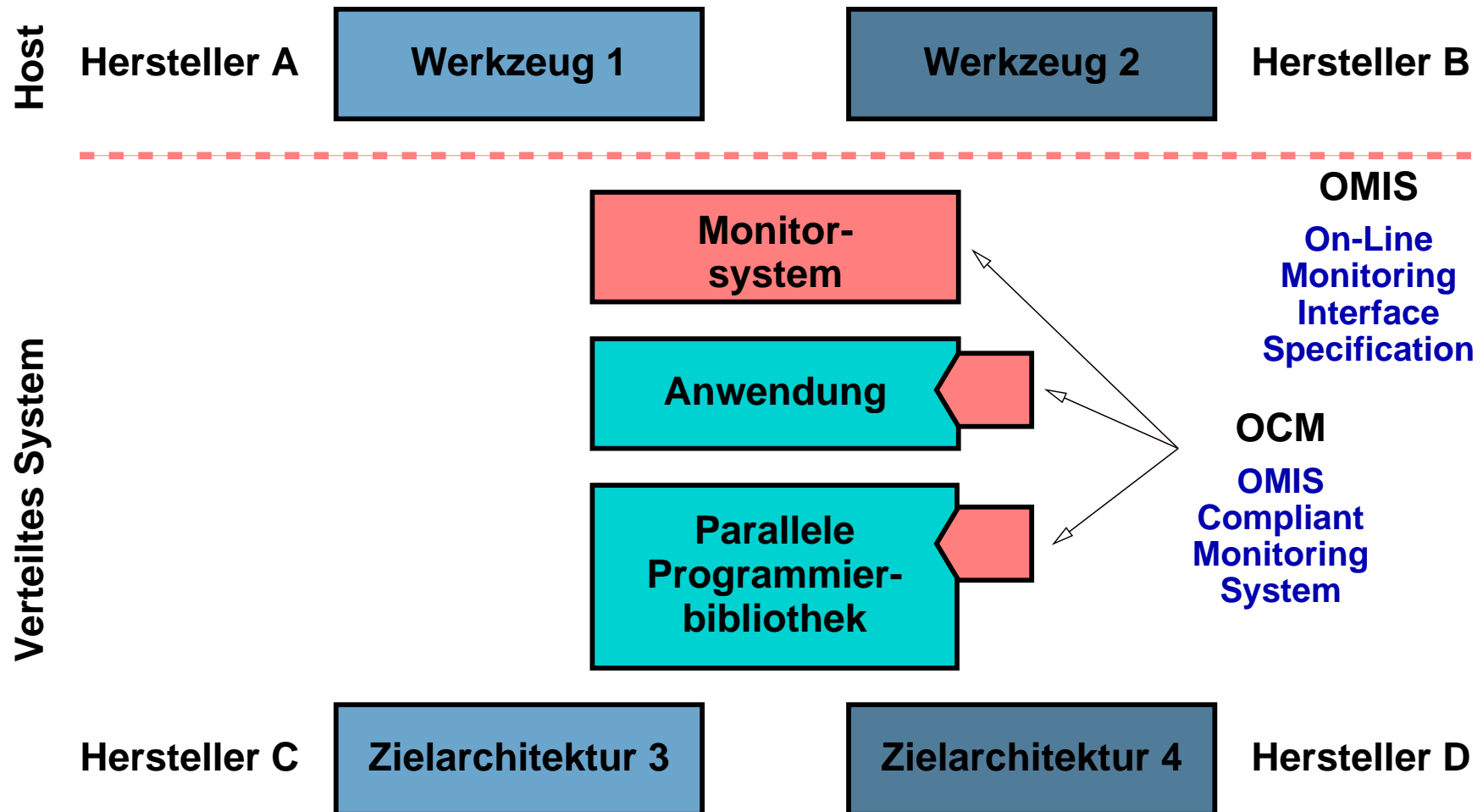
- ➔ Basis für Werkzeugintegration und Werkzeuginteroperabilität

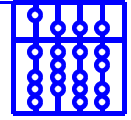
Gemeinsames Monitorsystem auf verschiedenen Architekturen

- ➔ Basis für universelle Werkzeuge



Schnittstellenbasierte Werkzeugkonstruktion





Modell der schnittstellenbasierten Werkzeugkonstruktion

OMIS 1.0 (On-Line Monitoring Interface Specification)

Konzepte zur Interoperabilität

OMIS 2.0, OMIS 3.0

Entwurf und Realisierung eines verteilten Monitorsystems

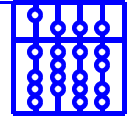
OCM (OMIS Compliant Monitoring System)

Entwurf und Realisierung interoperabler Online-Werkzeuge

Fehlersuche und Sicherungspunktgenerator

Übertragung der Konzepte auf andere Zielarchitekturen

OMIS 3.0, MiMo (Middleware Monitoring System)



Unterstützung von Interoperabilität und Universalität

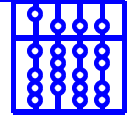
- ➔ Mehrere Werkzeuge nebenläufig anwendbar
- ➔ Werkzeuge auf mehreren Zielarchitekturen ablauffähig

Allgemeingültigkeit und Flexibilität

- ➔ Verschiedene Werkzeuge zu unterschiedlichen Zwecken
- ➔ Verschiedene Werkzeugarchitekturen
- ➔ Verschiedene programmiersprachliche Objekte

Erweiterbarkeit

- ➔ Neue Werkzeugkonzepte
- ➔ Neue zu überwachende Objekte



Objektklassen

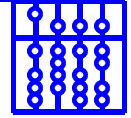
- ➔ System, Knoten
- ➔ Prozesse, Threads
- ➔ Nachrichtenpuffer, Nachrichten
- ➔ Monitorobjekte

Dienstklassen

- ➔ Informationsdienste (z.B. liefere Liste der Prozesse)
- ➔ Manipulationsdienste (z.B. halte Prozeß an)
- ➔ Benachrichtigungsdienste (z.B. melde, daß Prozeß terminiert)

Sicht des Werkzeugs

- ➔ Monitorsystem ist Server, der Dienste an Objekten anbietet

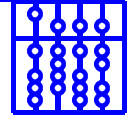


Arbeitsprinzip: Ereignis/Aktions-Modell

- ➔ **Ereignis** = interessierender Zustandsübergang im überwachten System
- ➔ **Aktion** = erwünschte Beobachtung oder Manipulation des System

Dienstanforderungen

- ➔ Verhalten des Monitorsystems ist durch Ereignis/Aktions-Relationen bestimmt
- ➔ Programmierung der Relationen über die definierte Schnittstelle
- ➔ Aktionen jeweils ausgelöst, wenn das Ereignis erkannt wird
- ➔ Leere Ereignisdefinition \Rightarrow unbedingte Aktion



Syntax der Schnittstellenfunktion

Omis_reply

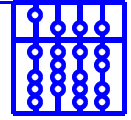
```
omis_request( char * request ,
              void (* callback)(Omis_reply reply, void *param),
              void *param,
              Omis_flags flags )
```

Syntax der Ereignis/Aktions-Relationen

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

Semantische Details

- ➔ Dienste arbeiten auf Einzelobjekten und Objektmengen
- ➔ Dienste arbeiten auf Objekten beliebiger Hierarchiestufen
- ➔ Aktionen können potentiell nebenläufig ausgeführt werden



Gewünschte Information: Verweilzeit im Sendeaufwurf und Anzahl gesendete Datenelemente

Was ist zu tun

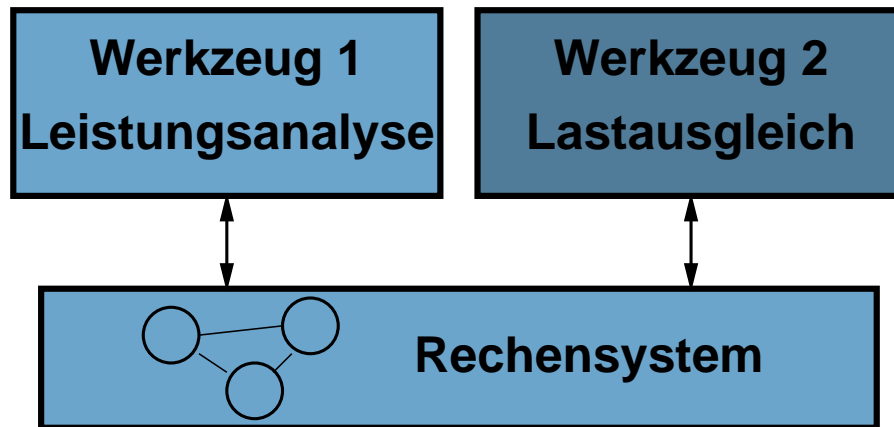
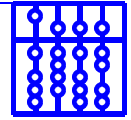
- ➔ Starten und Stoppen einer Stoppuhr
- ➔ Verwendung eines inkrementierenden Zählers

```
thread_has_started_lib_call([p_21],"MPI_Send") :
  pt_integrator_start(pt_i_1) pt_counter_add(pt_c_1,$par5)
```

- ☞ Wenn Prozeß p_21 einen Sendeaufwurf startet: aktiviere einen integrierenden Zähler und addiere die Anzahl Datenelemente (\$par5) auf einen Zähler

```
thread_has_ended_lib_call([p_21],"MPI_Send") : pt_integrator_stop(pt_i_1)
```

- ☞ Wenn der Prozeß den Aufruf beendet: halte den integrierenden Zähler an

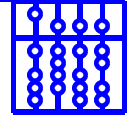


Problem

- ➔ Mehr als ein Werkzeug kennt den Zustand von O
- ➔ Mindestens ein Werkzeug verändert den Zustand von O
- ➔ Frage: Was machen die anderen Werkzeuge?

Notwendige Voraussetzung

- ➔ Einheitliche Schnittstelle für alle Werkzeuge
- ➔ Gemeinsames Monitorsystem



Zielvorstellung

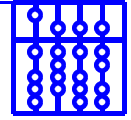
- ➔ Mehrere Werkzeuge können nebenläufig konsistent genutzt werden

Varianten in obigem Beispiel aus Werkzeugsicht

- ➔ Leistungsanalyse weiß nichts vom Lastausgleich
- ➔ Leistungsanalyse kennt Migration als Zustandswechsel

Varianten aus Benutzersicht

- ➔ Werkzeug visualisiert die Prozeßmigration
(Erwünschte Sicht eines Werkzeugentwicklers)
- ➔ Werkzeug verdeckt die Prozeßmigration
(Erwünschte Sicht eines Anwendungsprogrammierers)



Ausgangspunkt: Objektsicht des Systems

Frage: Wer greift wann wie auf Objekte zu?

Zugriffsarten: Lesend, modifizierend

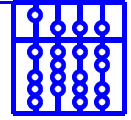
Zugriffskonflikt

☞ Inkompatibler zeitlich überlappender Zugriff zweier Werkzeuge auf ein gemeinsames Objekt O, wobei mindestens ein Zugriff modifizierend ist.

Möglichkeiten: RW, WR, WW

Mögliche Folgen von Zugriffskonflikten

- ➔ Abbruch des Programms
- ➔ Falsche Informationen im Werkzeug
- ➔ Abbruch des Werkzeuglaufs



Konsistenzproblem

- ➔ Werkzeug A bearbeitet ein Objekt, das von Werkzeug B später modifiziert wird.

Situationen: **R**W und **W**W

- ➔ Lösung: Werkzeug bittet um Benachrichtigung, wenn der Zustand eines Objekts verändert wird

Transparenzproblem

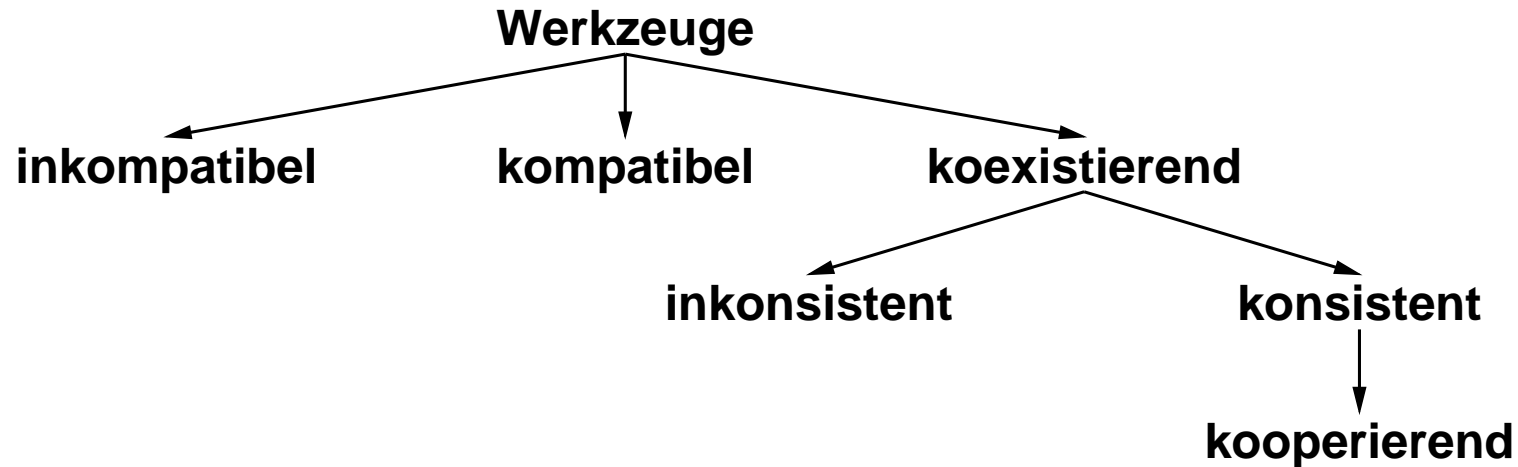
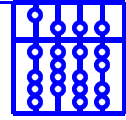
- ➔ Werkzeug B greift auf ein Objekt zu, das von Werkzeug A modifiziert wurde

Situationen: **W**R und **W**W

- ➔ Lösung: noch nicht endgültig erarbeitet

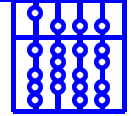
Transaktionsmechanismen und Sperren: zu hohe Zusatzlast

Abfangen von Zugriffen anderer Werkzeuge (eigene Modifikationen verdecken)



Beispiel: Leistungsanalyse / Lastausgleich

- ☞ Inkonsistent koexistierend: Leistungsanalysator ist verwirrt durch Verschwinden und Auftauchen von Prozessen
- ☞ Konsistent koexistierend: Leistungsanalysator erkennt Verschiebungen und visualisiert sie falls gewünscht. Ansonsten verdeckt er sie
- ☞ Kooperierend: Leistungsanalysator gibt selber Anstoß zur Lastverwaltung; Lastverwalter beauftragt Leistungsanalysator mit der Visualisierung der Verwaltungseffekte



PCTE — Portable Common Tool Environment (Esprit 1983)

- ➔ Interoperabilität bei Software-Entwurfswerkzeugen

ToolTalk (Sun Microsystems)

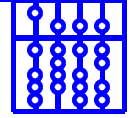
- ➔ Interoperabilität für Programme unter X Windows

DAMS (Universidade Nova de Lisboa, Lissabon, Portugal)

- ➔ Einheitliches Monitorsystem mit Ansätzen zu Interoperabilität
keine Konfliktauflösung

DPCL — Dynamic Probe Class Library (IBM u.a.)

- ➔ Monitorschnittstelle ohne Konzepte zur Interoperabilität

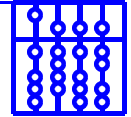


Verteilter gemeinsamer Speicher

- ➔ Hier wichtig: Adreßraumbereiche
- ➔ Erweiterung von OMIS um die Objektklasse „Speicherbereich“
- ➔ Erweiterung von OMIS um neue Dienste zu dieser Objektklasse
- ➔ Anpassung des Monitorsystems an diesen Dienste

CORBA-basierte Programmsysteme

- ➔ Übertragung der Schnittstellenkonzepte in den Bereich der Middleware-Systeme
- ➔ Adaption einzelner Komponenten an CORBA



Modell der schnittstellenbasierten Werkzeugkonstruktion

1996: OMIS 1.0

Konzepte zur Interoperabilität

1997: OMIS 2.0

Frühjahr 2000: OMIS 3.0

Entwurf und Realisierung eines verteilten Monitorsystems

1997-1999 OCM

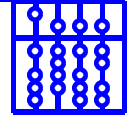
Entwurf und Realisierung interoperabler Online-Werkzeuge

Herbst 1999: Fehlersuche und Sicherungspunktgenerator

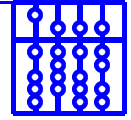
Übertragung der Konzepte auf andere Zielarchitekturen

OMIS 3.0

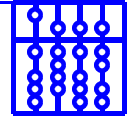
Sommer 2000: MiMo (Middleware Monitoring System)



- ☞ Institute of Computer Science and Academic Computer Center
CYFRONET, Krakau, Polen
Anpassung des Leistungsmeßwerkzeugs PATOP an OCM; Adaption des OCM an das Message Passing Interface (MPI)
- ☞ Universität Wien
Verwendung des MPI-OCM aus Krakau für ein Fehlersuchwerkzeug für High Performance Fortran (HPF)
- ☞ KFKI-MSZKI Research Institute, Budapest, Ungarn
Verwendung des OCM in eigener Entwicklungsumgebung
- ☞ University of Wisconsin, Madison, Wisconsin, USA
Einbindung von lowlevel-Routinen zur Systemmanipulation in den OCM
- ☞ Argonne National Laboratories, Illinois, USA
Einbringung von Monitor Konzepten in CORBA und Globus



- ➡ Bildgebende Verfahren in der Computertomographie (PET, fMR)
Vergleich verschiedener Programmiermodelle und Integration von dynamischem Lastausgleich in die Anwendungsprogramme
- ➡ SEEDS: Verteilter Simulator für die Bewertung von Flughafenbodenverkehr
Evaluierung einer geeigneten Hardware-/Software-Architektur für das verteilte System
- ➡ Kooperation mit BMW: Parallelisierung von Pamcrash
Vergleich verschiedener Programmiermodelle und Integration von dynamischem Lastausgleich
Kontrolle des Nichtdeterminismus durch geeignete Online-Werkzeuge (Deterministische Programmausführung)



- ➡ Eine effiziente Methode zur Konstruktion von Online-Werkzeugen stützt sich auf die Spezifikation einer universellen Schnittstelle ab
- ➡ Ausgehend von einer gemeinsamen Schnittstelle sind erstmals Konzepte der Werkzeuginteroperabilität unterstützbar
- ➡ Die neu entwickelten Werkzeuge haben einen besseren Funktionsumfang und ermöglichen potentiell Synergieeffekte
- ➡ Es entstehen dabei viele allgemeine Konzepte, die dem Bereich der Betriebssysteme und verteilten Systeme zuzuordnen sind
- ➡ Künftige Arbeiten konzentrieren sich auf den Bereich verschiedener konkreter interoperabler Werkzeuge und die Übertragung der gewonnenen Erkenntnisse auf andere Systemarchitekturen