



## Einsatzbereiche

- Entwicklung, Optimierung und Laufzeitunterstützung paralleler und verteilter Programme

## Benutzungsmethodik

- Interaktiv — Automatisch

## Benutzungszeitpunkt

- Zur Laufzeit (online) — Nach Programmende (offline)

## Programminteraktion

- Beobachtung — Manipulation

## Programmiermodell

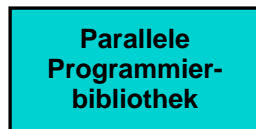
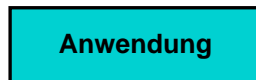
- Nachrichtenaustausch zwischen Prozessen



Hersteller A



Hersteller B



Verschiedene Werkzeuge arbeiten kooperierend auf demselben Programm

# Methoden zur Realisierung interoperabler Werkzeuge zur parallelen Programmierung

Thomas Ludwig

Institut für Informatik, Technische Universität München  
 Institut für Informatik, Ludwig-Maximilians-Universität München  
 e-mail: ludwig@informatik.tu-muenchen.de



- Software-Werkzeuge
- Interoperable und universelle Werkzeuge
- Konventioneller Werkzeugentwurf
- Methodik des effizienten Werkzeugentwurfs
- Spezifikation der Monitorschnittstelle
- Arbeitsprinzipien und Schnittstellenfunktionen
- Anwendungsbeispiele
- Entwurf eines Monitorsystems
- Ausblick

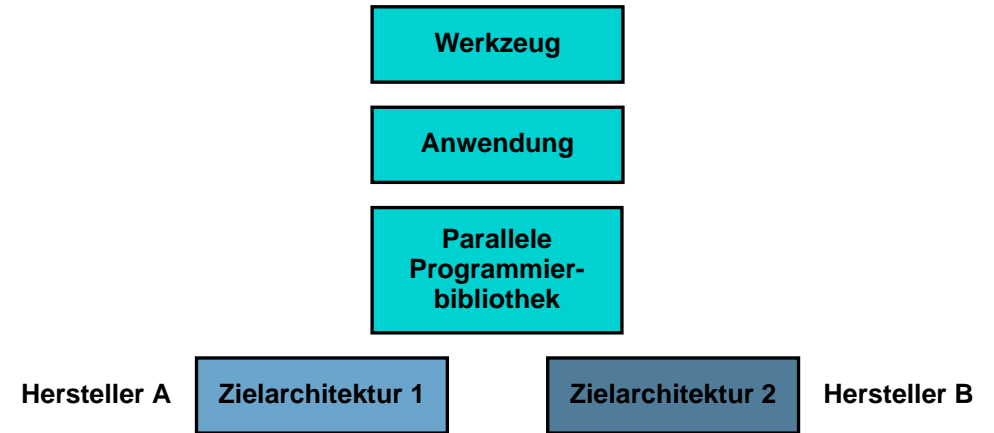
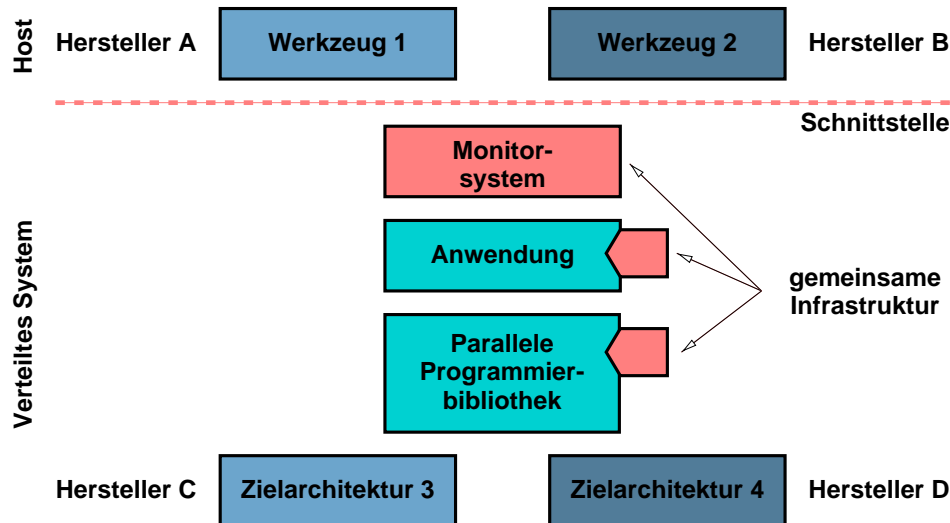


## Vorgehensweise

1. Abtrennen der Steuerkomponente des Werkzeugs von der Werkzeuginfrastruktur
2. Einführung einer genormten Schnittstelle zwischen beiden Teilen
3. Identifikation von Infrastrukturtteilen, die vielen Werkzeugen gemeinsam sind
4. Zusammenführen gemeinsamer Komponenten in einer einheitlichen Implementierung

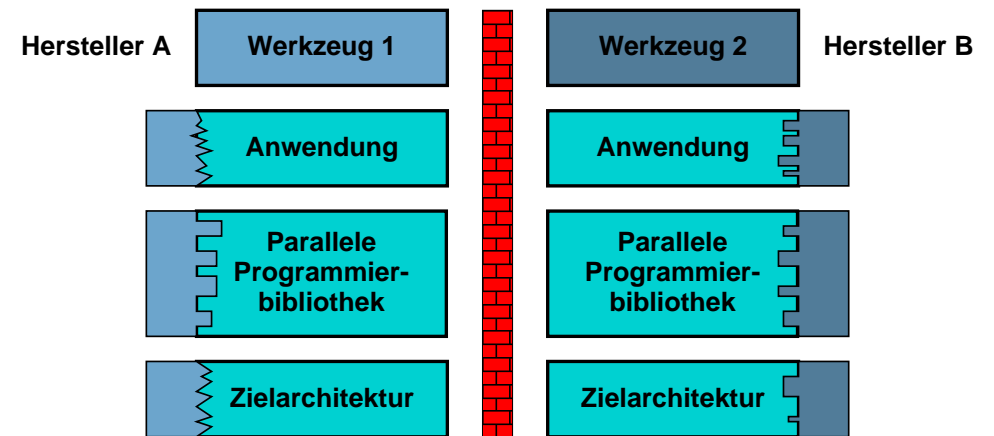
⇒ Schnittstellen + Infrastrukturmodule

# Effizienter Werkzeugentwurf

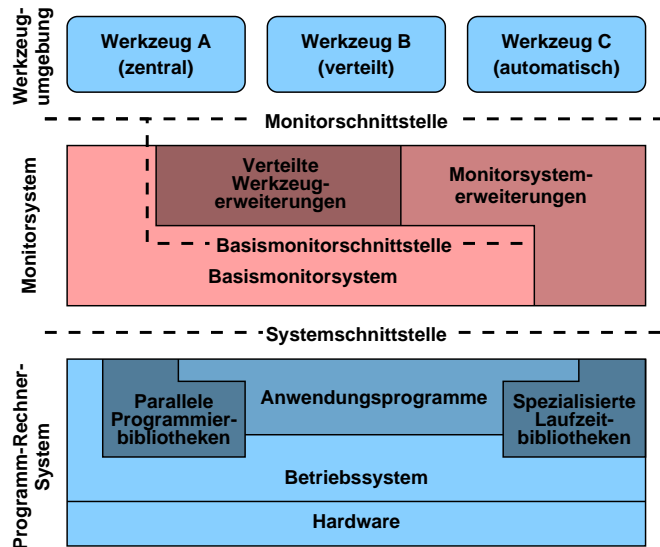


Dasselbe Werkzeug arbeitet auf verschiedenen Zielarchitekturen

# Konventioneller Werkzeugentwurf



Werkzeuge sind in Implementierung und Anwendung miteinander unvereinbar



## Aus Sicht des Werkzeugs:

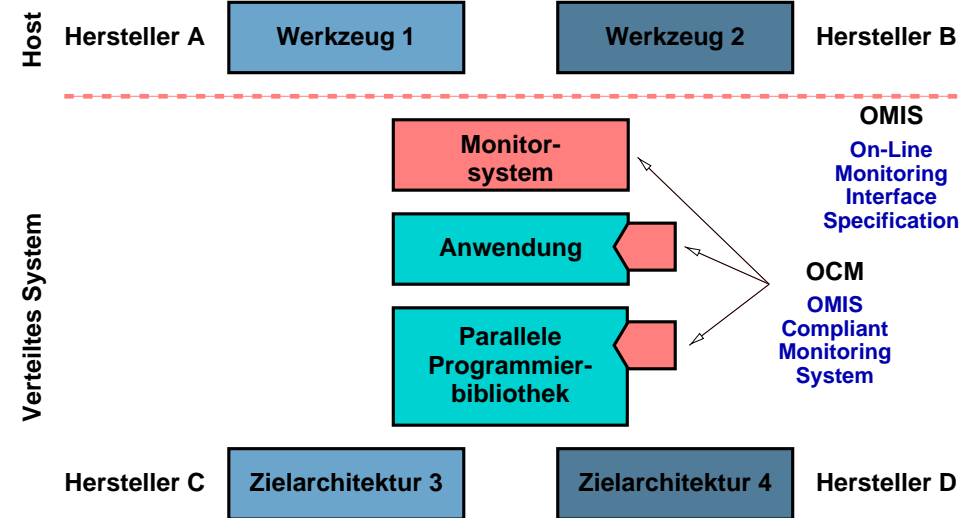
- Monitorsystem ist Server, der Dienste an Objekten anbietet

## Dienstklassen:

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

## Objektklassen

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



## 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

## Effizienz

- Geringes Kommunikationsaufkommen zwischen Werkzeugen und Überwachungssystem



### Arbeitsprinzip: Ereignis/Aktions-Modell

- **Ereignis** = interessierender Zustandsübergang im überwachten Programm
- **Aktion** = erwünschte Beobachtung oder Manipulation des Programms

### 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



### Knoten

- Statische / dynamische Informationen
- Überwachung einrichten / aufgeben, ...

### Prozesse

- Statische / dynamische Informationen
- Erzeugung, Überwachung einrichten / aufgeben, Modifikation des Speichers, Ändern der Priorität, ...
- Terminierung, Empfang eines Signals, ...

### Threads

- Statische / dynamische Informationen
- Überwachung aufgeben, Modifikation von Registern, Anhalten, Weiterführen, ...
- Erzeugung von Prozessen / Threads, Hinzufügung / Löschung von Knoten, Ausführung von Bibliotheksfunktionen, ...



### Nachrichtenpuffer und Nachrichten

- Inhalt des Puffers, Sender einer Nachricht, ...
- Nachricht aus Puffer löschen, Nachricht markieren, ...
- Eintrag einer Nachricht in einen Puffer, Empfang einer markierten Nachricht, ...

### Monitorobjekte

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



## Unterstützung von Interoperabilität

- Einstellbarer Wirkungsbereich jedes Werkzeugs
- Exklusives Ausführen von Aktionslisten
- Geeignete Wahl der Benachrichtigungsdienste

## Unterstützung von Universalität

- Zielarchitektur an der Schnittstelle nicht sichtbar
- Wesentlich unterstützt durch Implementierungskonzepte



## Koppelung automatischer und interaktiver Werkzeuge

### Fehlersuche und Sicherungspunktgenerierung

- Abspeichern von Programm- und Sitzungszustand
- Wiederaufsetzen der Sitzung von Sicherungspunkten aus
- Aufnahme einer Fehlersuchung von Sicherungspunkten aus

### Leistungsanalyse und Lastausgleich

- Bewertung der Programmleistung unter Produktionsbedingungen
- Erfassung der Leistung des Lastverwaltungssystems
- Einblenden/Ausblenden des Wirkens der Lastverwaltung



```
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



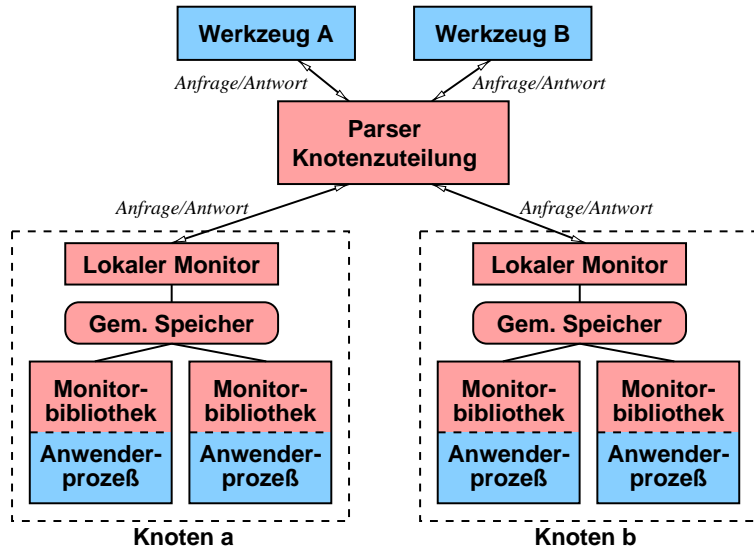
```
thread_has_started_lib_call([p_21], "MPI_Send") :
pt_integrator_start(pt_i_1) pt_counter_add(pt_c_1, $par5)
```

Wenn Prozeß `p_21` einen Sendeaufruf startet: aktiviere einen integrierenden Zähler und addiere die Nachrichtenlänge (`$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

**Resultat:** Verweilzeit im Sendeaufruf und gesendete Nachrichtenmenge



## OMIS

- Version 1.0 veröffentlicht Februar 1996
- Überarbeitung: Version 2.0 veröffentlicht Juli 1997

## OCM

- Implementierung begonnen Januar 1997
- Erster Prototyp: Juli 1997
- Erste Werkzeuge des TOOL-SET: Oktober 1997

## Kooperationen

- ENS Lyon, France (Dosmos)
- KFKI-MSZKI Research Institute, Budapest, Hungary (GRADE)
- GUP, Universität Linz, Österreich (MAD)



## Implementierung für PVM-basierte Anwendungen auf Rechnernetzen

### Projektziele:

- Verifikation der Anwendbarkeit der Spezifikation
- Implementierungsplattform für THE TOOL-SET
- Implementierungsplattform für Werkzeuge Dritter
- Referenz für andere OMIS-konforme Implementierungen



- Das Monitorsystem wird aus kommunizierenden Monitoren gebildet, jeweils einem pro Rechnerknoten der virtuellen Maschine
- Globale Aktionsspezifikationen werden zur Definitionszeit an die Knoten verteilt.
- Kommunikation zwischen Monitoren mittels PVM-Mechanismen
- Im ersten Schritt: Zentrales Parsen der Dienstanforderungen
- Ereignisgetriebenes Ausführungsmodell der (Einzelknoten-)Monitore
- Ereigniserkennung und Aktionsausführung sowohl im Monitor als auch in der Anwendung
- Kommunikation zwischen knotenlokalen Monitorkomponenten über gemeinsamen Speicher und Signale



- Abschluß der OCM-Implementierung und Werkzeuganpassung
- Optimierung der OCM-Implementierung
- Integration neuer Werkzeuge: Sicherungspunkterstellung, Lastausgleich, deterministische Programmausführung
- OMIS für Architekturen mit verteiltem gemeinsamen Speicher
- Kooperationen mit anderen Forschungsgruppen

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