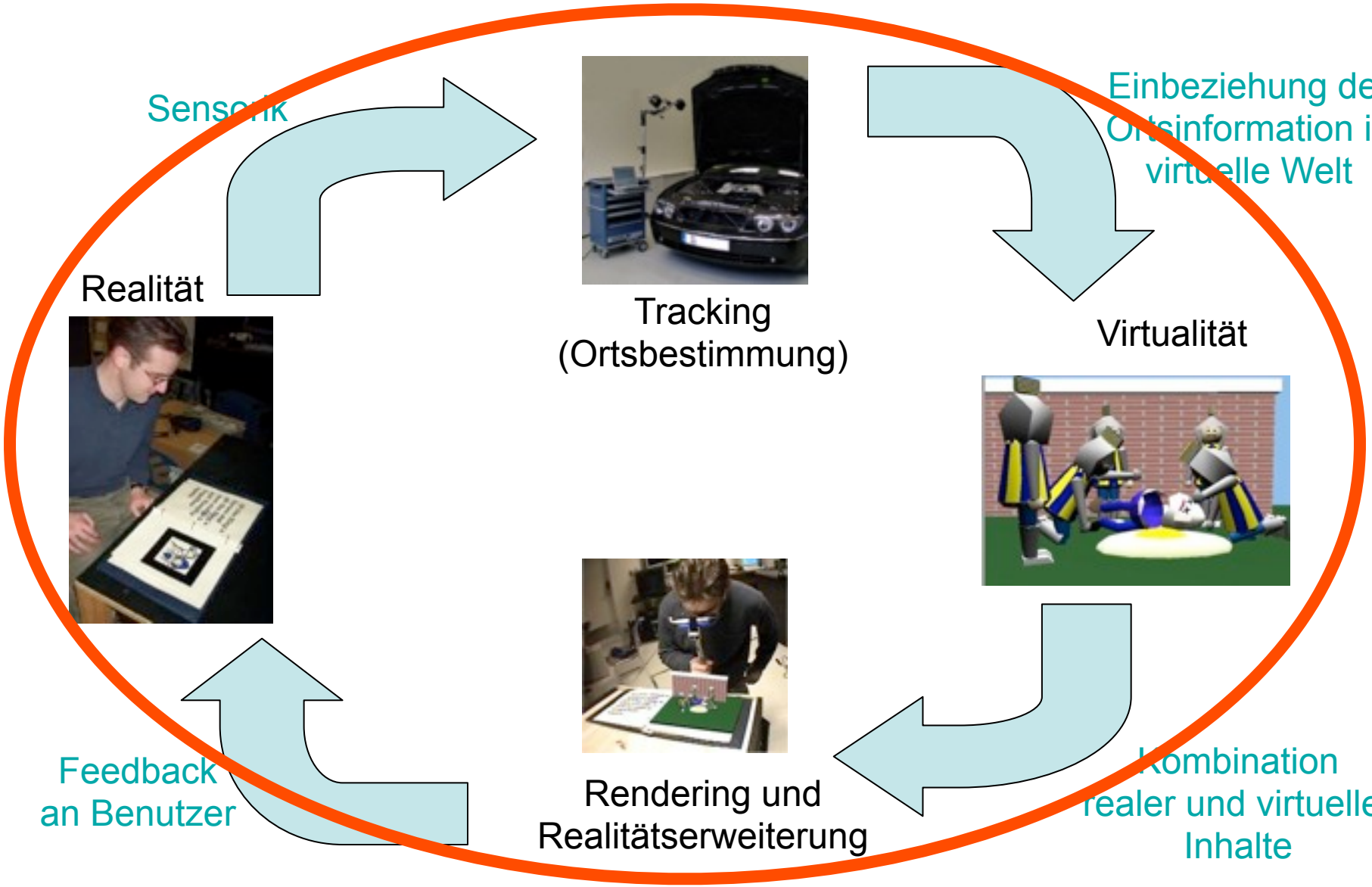
Softwaresysteme für Augmented Reality

Vorlesung „Augmented Reality”

Andreas Butz

Ein Generisches AR-System



Überblick

- Architekturstile
  - Monolithische Systeme
  - Komponentenbasierte Systeme
    - Client/Server
    - Peer-to-Peer
    - Blackboard
    - Pipe and Filter
- Verteilte Systeme: Kommunikationsarten
  - RPC/RMI
  - Events
  - Shared Memory

Architectural Style

A style consists of a vocabulary of design elements, a set of well-formedness constraints that must be satisfied by any architecture written in the style, and a semantic interpretation of the connectors.
[Moriconi 1994]

- Beschreibt allgemeine Lösung des Systemaufbaus
- Wird während des Systemdesigns festgelegt
- Domänenunabhängig
- Abstraktester Beschreibungsteil einer Architektur
- Reicht nicht aus, um eine Architektur zu beschreiben!

Arch. Style: Monolithische Systeme

- Alle Systemteile im selben Programm
- Standardansatz von „Hacks“
- Beispiele:
  - (Alte) Mainframeprogramme
  - Viele DOS/Windows 3.x Anwendungen
  - Projekte im Übungsbetrieb AR

Arch. Style: Diskussion Monolithe

Pro:

- Schnell programmiert (aber nur für sehr einfache Probleme)
- Einfaches Einmal-Deployment

Con:

- Schwierige Wartbarkeit
- Sehr schlechte Wiederverwendung
- Nebenläufigkeit schwer zu realisieren

Don‘t try this at home.

Komponentenbasierte Architekturen

- Idee: Baue Programme aus fertig fabrizierten Komponenten (Douglas McIlroy, *Mass Produced Software Components*, 1968)
- Häufige Implementierung:
  - Komponenten sind Objekte
  - Definierte Schnittstellen in einer Interface Description Language (IDL)
- Komponenten können auf einen (z.B. Microsoft COM) oder mehrere Rechner (z.B. CORBA, DCOM) verteilt werden

Arch. Style: Client/Server

- Ein oder mehrere Server bieten Dienste für einen oder mehrere Clients
- Client ruft bekannte Schnittstelle des Servers auf und erhält Antwort
- Benutzer interagieren nur mit dem Client
- Beispiele
  - Webserver und Browser
  - NFS-Server und –Clients
  - Mailserver und Mailclient

Arch. Style: Diskussion Client/Server

Pro:

- Einfache Struktur
- Zentralisierte Datenverwaltung
- (Tragbarer) Client kann mit geringer Rechenleistung auskommen

Con:

- Hohe Abhängigkeit von Server (Redundanz nötig)
- Besonders für Bildverarbeitung hohe Netzwerklast
- Existierende Systeme nicht auf Echtzeit ausgelegt (eher auf interaktive Bedienung)

Arch. Style: Peer to Peer (P2P)

- Jede beteiligte Komponente ist zugleich Server und Client
- Mischformen möglich, in denen Server, Clients und (Server+Client)s existieren
- Oft zusammen mit Ad-hoc Verbindungen
- Beispiele:
  - Filesharing Netze
  - Standard-Arbeitsgruppe in Windows-Netzwerk
  - Internet (konzeptionell)

Arch. Style: Diskussion P2P

Pro:

- Höchste Flexibilität
- Extreme Ausfallsicherheit möglich
- Passt gut in das Paradigma des Ubiquitous Computing

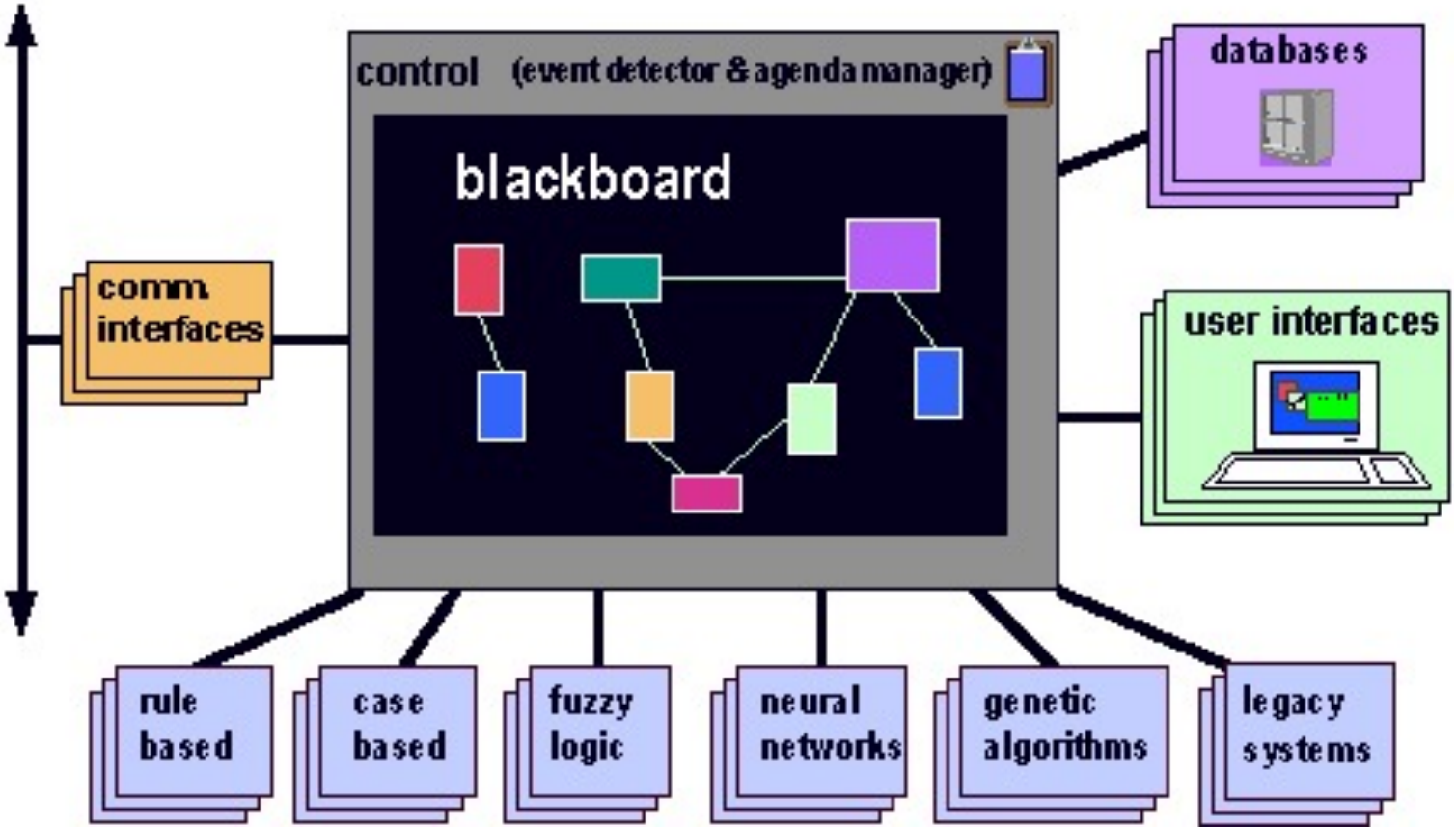
Con:

- Sehr komplex
- Unbekannte Struktur muss antizipiert werden
- Rekonfigurationen zur Laufzeit nötig (bei Ad-hoc Verbindungen)

Arch. Style: Blackboard

- „Schwarzes Brett“ als zentrales Repository
- Tuple Spaces als theoretische Basis
  - In Blackboard werden Tupel gelegt (Menge von Name/Wert Paaren)
  - Abfrage durch partielle Tupel (gegebene Name/Wert Paare müssen matchen, die restlichen werden ausgefüllt)
- Beispiele:
  - Agentensysteme in der KI
  - Java Spaces

Arch. Style: Blackboard Bsp.



[http://www.nb.net/~javadoug/bb.htm]

Arch. Style: Diskussion Blackboard

Pro:

- Sehr lose Kopplung von Komponenten
- Einfaches Multiplexing von Events
- Sehr intuitive Bedienung

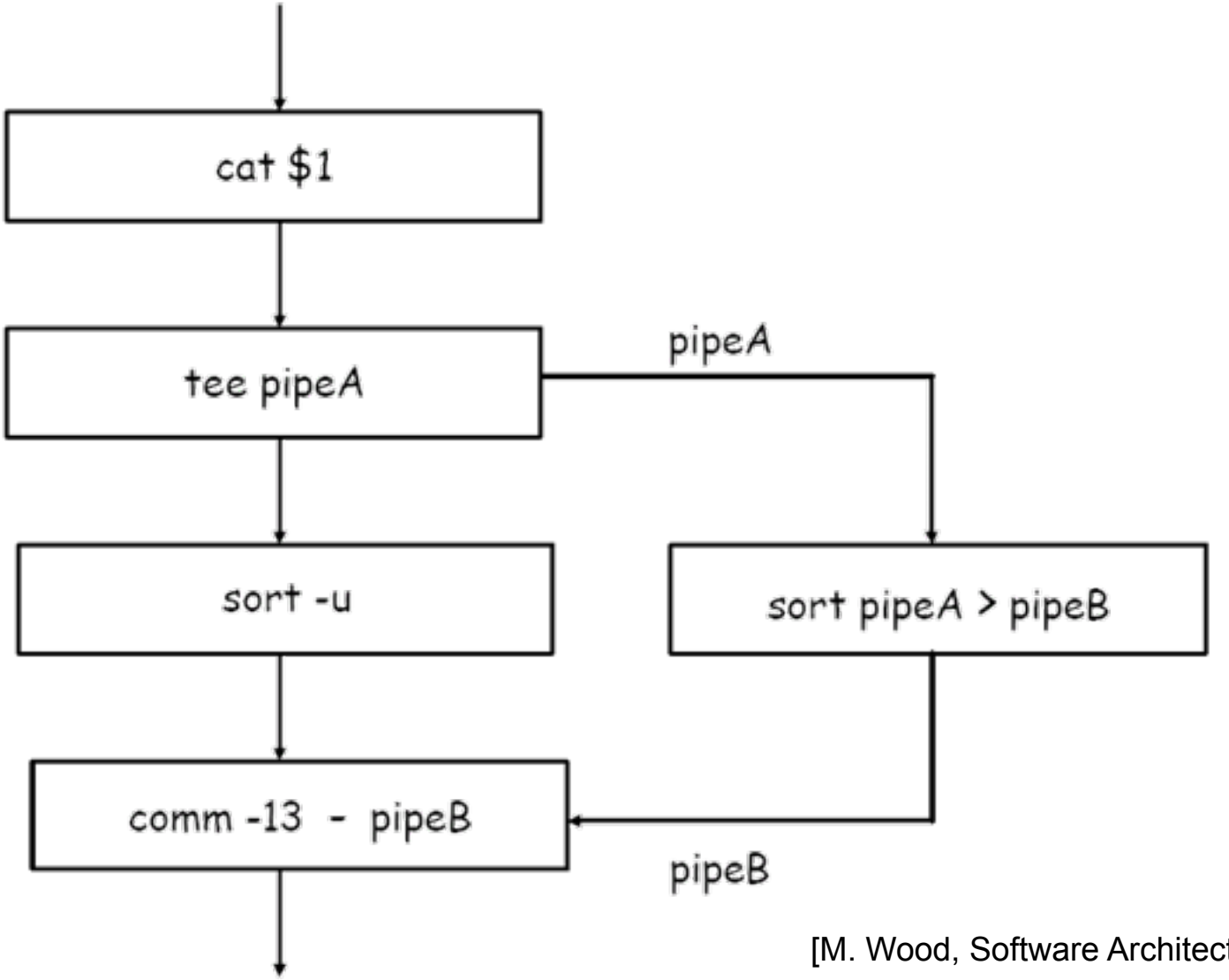
Con:

- Zentrale Komponente (Ausfallprobleme)
- Relativ ineffizient
- Hoher Ressourcenbedarf

Arch. Style: Pipe and Filter

- Datenflussarchitektur
- Jede Komponente liest Datenstrom als Eingabe und produziert Datenstrom als Ausgabe (→ Filterkomponenten)
- Verbindungen zw. Filtern durch Pipes
- Beispiele:
  - UNIX Shell-Skripte
  - Microsoft DirectShow Framework

Arch. Style: Pipe and Filter Bsp.



[M. Wood, Software Architecture Class]

Arch. Style: Diskussion Pipe and Filter

Pro:

- Verständlichkeit
- Hohe Wiederver-wendbarkeit der Komponenten
- Erweiterbarkeit (neue Filter) und Verbesserbarkeit (Filter ersetzen)
- Nebenläufigkeit einfach modellierbar

Con:

- Laufzeit kann recht hoch werden
- Keine globale Systemsicht – ein Filter kennt nur sich selbst
- Wie setzt man das Filternetzwerk (automatisch) auf?

Kommunikation in Verteilten Systemen

- Problem: wie kommunizieren (potentiell über ein Netzwerk verteilte) Komponenten
- Ziele:
  - Einfache API
  - Hohe Performanz
  - Gute Skalierbarkeit

RPC/RMI

- RPC = Remote Procedure Call, bei objektorientierten Systemen oft auch RMI = Remote Method Invocation
- Scheinbar lokaler, direkter Aufruf einer (entfernten) Komponente, die nur durch eine IDL spezifiziert ist
- Folgt Client-Server Modell
- Beispiele:
  - Microsoft DCOM
  - CORBA
  - XML-RPC / SOAP (Web Services: http als Protokoll, IDL in XML spezifiziert)

RPC/RMI: Diskussion

Pro:

- Nach Verbindungs-aufbau einfache Kommunikation (wie unter lokalen Komponenten)
- Hohe Performanz
- Gut für statusbehaftete Komponenten

Con:

- Multiplexing/ Broadcasting sehr schwierig
- Begrenzte Nachrichtengröße und –anzahl

Event-Kommunikation

- Grundidee: Programme warten auf Events, machen etwas, warten wieder
- Einsatz vor allem in GUI-Anwendungen → initiale Events kommen vom Benutzer
- Implementierung durch zentrale Event-Loop:

```
while (true) {
        waitForNextEvent();
        processEvent();
}
```

- Häufig: Kaskaden von Events

Events: Diskussion

Pro:

- Hervorragend für interaktive Anwendungen (GUIs)
- Natürliche Unterstützung von Nebenläufigkeit
- Multiplexing von Events problemlos (v.a. durch Broad- und Multicasts über UDP)

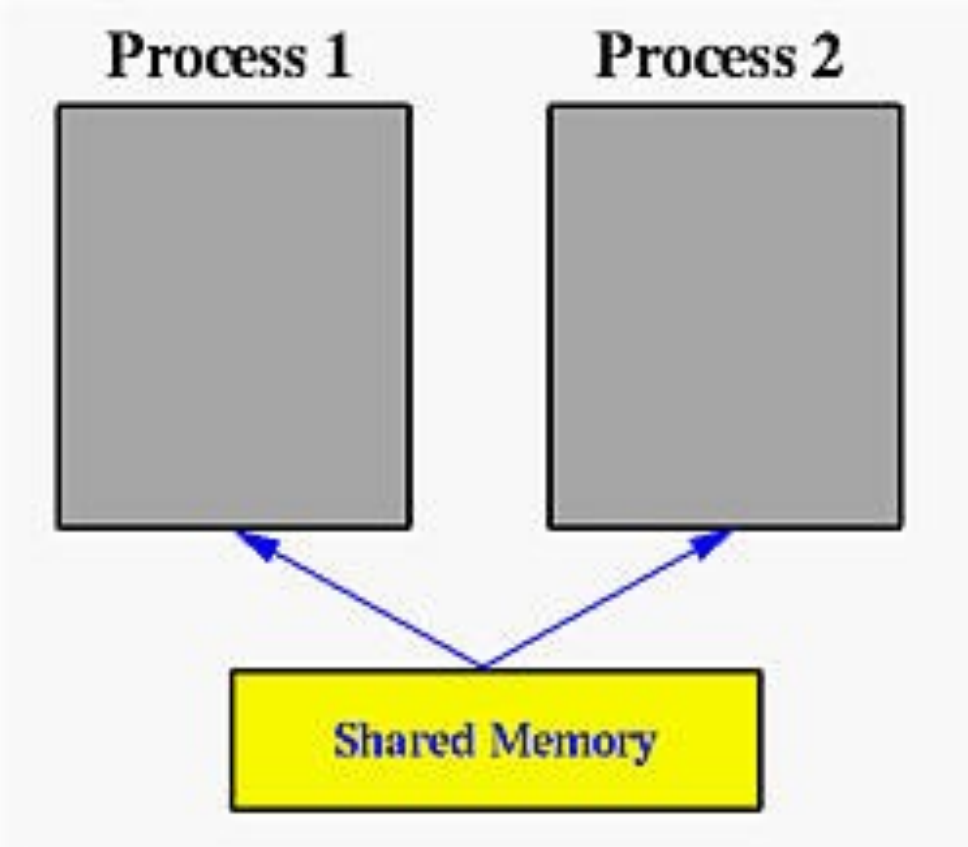
Con:

- Komponenten sollten zustandslos sein (v.a. bei unzuverlässiger Eventzustellung)
- Events können sich stauen, dann starke Verzögerungen in Abarbeitung möglich
- I.a. keine Garantien für Bearbeitungszeit eines Events möglich

Shared Memory

- Grundidee: Mehrere Prozesse teilen sich gemeinsamen Speicherbereich
- Vor allem für sehr breitbandige Übertragungen geeignet (Videodaten!)
- Analogie zur Blackboard Architektur
- Beispiele:
  - Multithreading: mehrere Threads greifen auf gemeinsamen Speicher zu
  - Parallelrechner (SMP): mehrere CPUs greifen auf gemeinsamen Speicher zu

Shared Memory: Bsp.



[http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/shm/what-is-shm.html]

Shared Memory: Diskussion

Pro:

- Höchst effizient
- Quasi beliebige Datengröße
- Beliebige Daten können ausgetauscht werden

Con:

- Extrem enge Kopplung
- Nur lokal möglich
- Synchronisation (z.B. über Semaphore) zwingend nötig

Beispielsysteme

- AR Toolkit
- VRIB/ VARIO
- Studierstube
- COTERIE/ Repo-3D

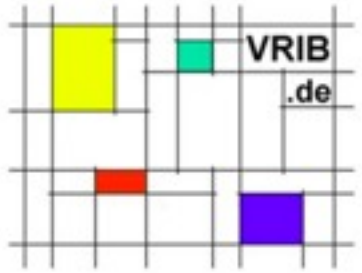
AR Toolkit

- Entwickelt am HITLab (Seattle, WA, USA)
- C-Bibliothek für Windows, Mac OS X, UNIX
- Bestandteile:
  - Videograbbing (DirectShow, Quicktime, V4L)
  - Markererkennung und -training
  - Hilfsfunktionen für OpenGL-Rendering auf dem Marker
  - Kalibrierungsroutinen

AR Toolkit: Diskussion

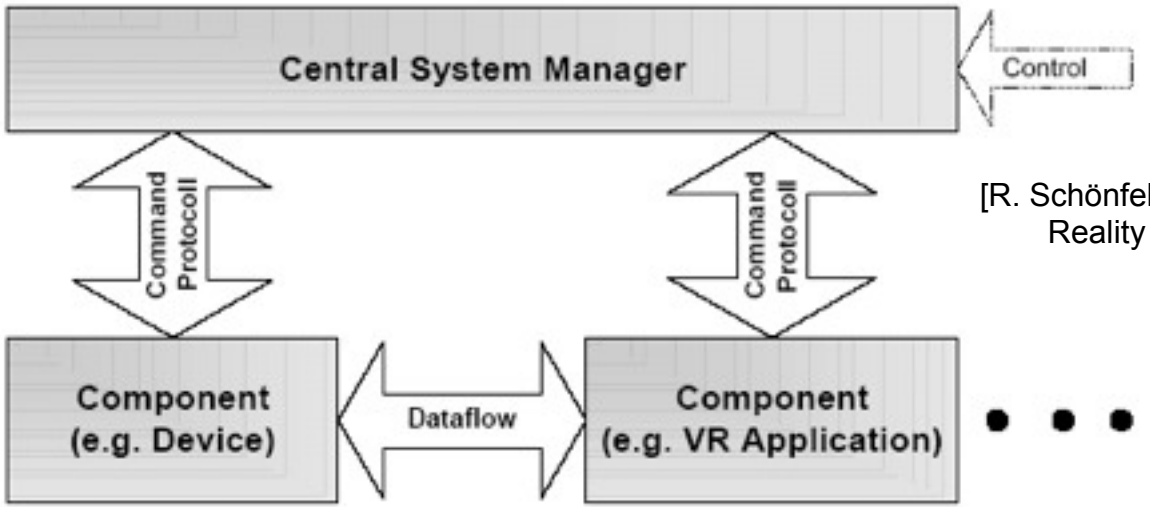
- Monolithisches System
- Läuft auf jedem halbwegs aktuellen Rechner (+ billige Webcam)
- Fokus auf markerbasiertem Tracking
  - Rendering nur direkt auf Markern genau
- Gut für schnelles Ausprobieren neuer AR-Interaktionen

VRIB/ VARIO



- BMBF-Projekt: *Virtual Reality Interaktionsbaukasten* (u.a. TU Ilmenau, DaimlerChrysler R&D)
- Ziel: „*Optimale Unterstützung der Methodik zu Entwurf und Umsetzung von 3D User Interfaces*“, also Rapid Prototyping von 3D User Interfaces
- VARIO (*Virtual and Augmented Reality Inputs and Outputs*) ist verteilte Softwarebasis von VRIB

VARIO: Architektur

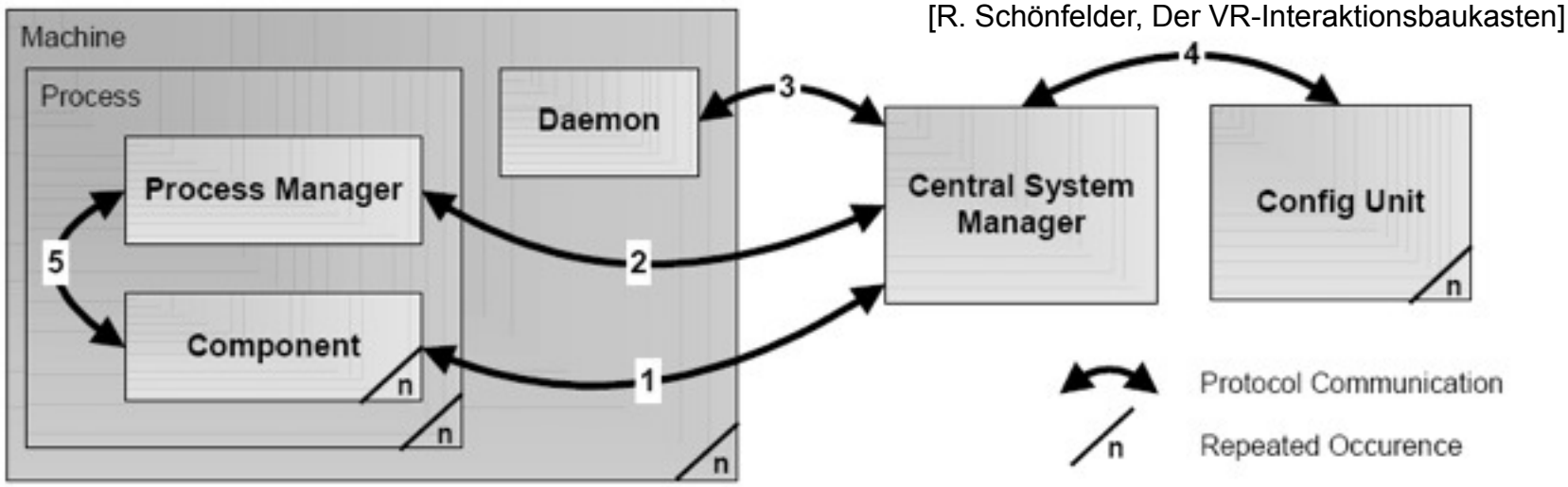


[R. Schönfelder, VRIB – The Virtual Reality Interaction Toolkit]

- Datenflussarchitektur (Pipe and Filter), von Interaktionsgerät hin zur Anwendung
- Zentrale Steuerung durch Central System Manager (CSM)
- Benutzer rekonfiguriert System zur Laufzeit über GUI für CSM
- Persistenz durch Speicherung des Systemzustands im XML-Format

VARIO: Architektur (2)

[R. Schönfelder, Der VR-Interaktionsbaukasten]

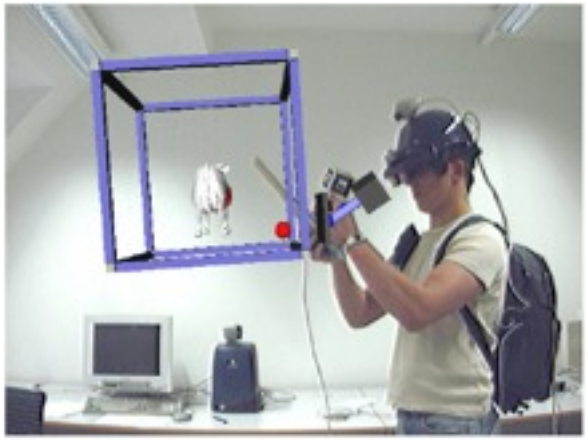


- Auf jedem Host kommuniziert ein *Daemon* mit dem CSM, dieser kann neue *Prozesse* starten
- Ein *Process Manager* kontrolliert je Prozess mehrere Komponenten
- Kommunikation über XML via TCP (Steuerung) und UDP-Streams (Datenfluss)

VARIO: Diskussion

- Zentrale Steuerung erlaubt sehr leichtes Management der Komponenten (GUI)
- Trotzdem hohe Laufzeiteffizienz durch Datenflussarchitektur mit UDP-Streams
- Laufzeitrekonfiguration ermöglicht Rapid Prototyping neuer Interaktionsideen
- Leider nicht öffentlich erhältlich…

Studierstube



- AR-Framework der TU Wien/Graz (Prof. Schmalstieg)
- Grundlage: Szenengraphbibliothek
- Grundidee:
  - Rendering ist wesentlicher Teil jeder AR-Anwendung
  - Anwendungslogik ist in Knoten des Szenegraphen codiert → wird bei Traversierung des Graphen automatisch ausgeführt

Studierstube: Diskussion

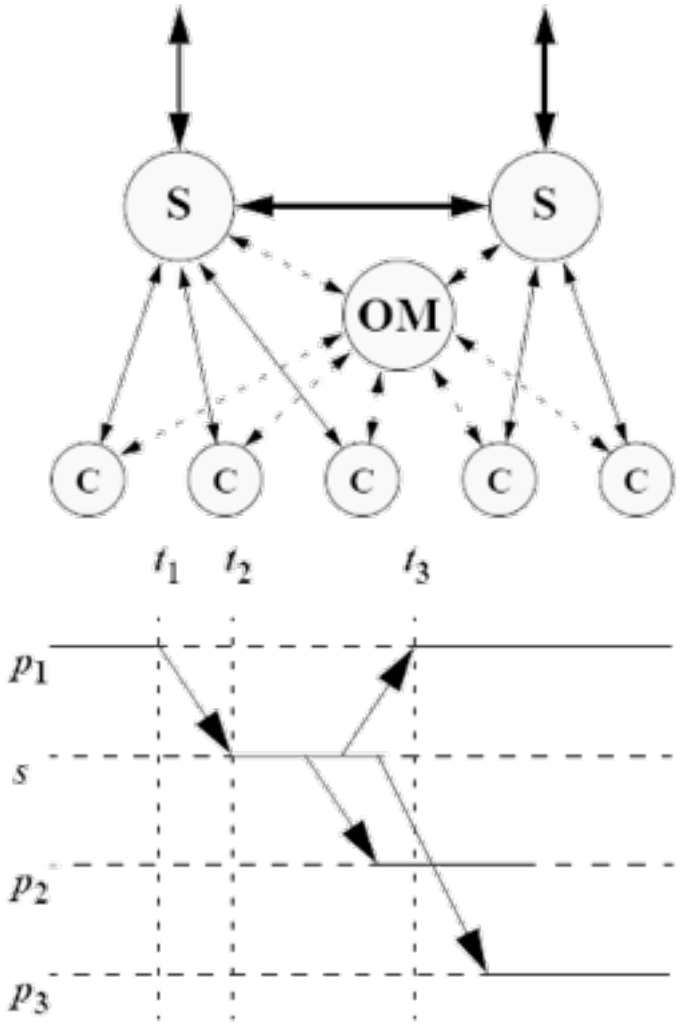
- Starker Fokus auf Interaktionstechniken für 3D-Umgebungen
  - Besondere Betonung des Renderings
- Verteilung:
  - Beim Tracking sehr flexibel, aber statisch
  - Für Anwendungen zur Laufzeit möglich, skaliert allerdings nicht besonders gut (Szenengraph wird voll repliziert)
- Extrem viele Anwendungen und Erweiterungen (u.a. Scripting)
- Frei erhältlich, auch im Source Code

COTERIE/ Repo-3D

- B. MacIntyre, Columbia University, 1996, „Columbia Object-oriented Testbed for Exploratory Research in Interactive Environments“
- Cross-Platform Library, Sprache: Modula 3 (Java hat viele der M3-Konzepte übernommen)
- Scriptsprache: Obliq + Repo (Replicated Obliq)
- Grundidee:
  - Verteilter Shared Memory Ansatz (ähnlich Blackboard)
  - Objektorientierung und Multithreading als Grundparadigmen (Kapselung)
- 3D Graphikbibliothek Repo-3D baut auf COTERIE

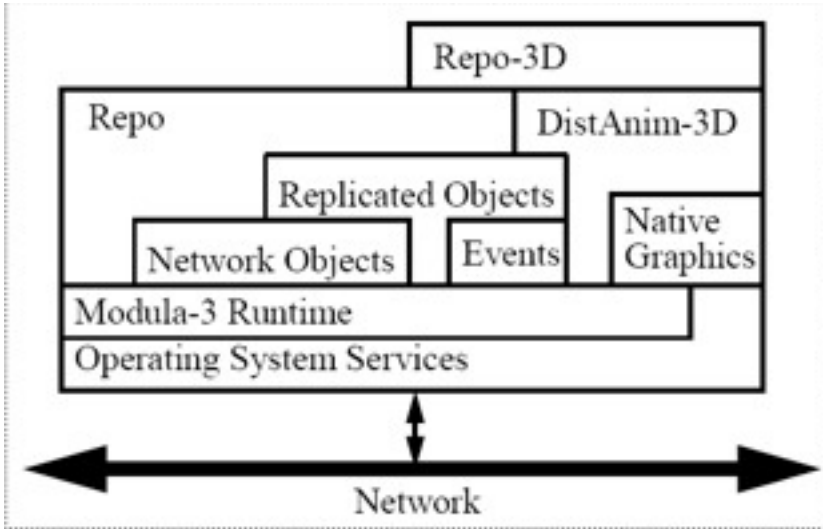
COTERIE: Verteilungsarchitektur

- Hauptproblem: wie stellt man Synchronisation sicher?
- Sequencer empfängt Event, sobald ein Objekt geändert wird
- Dann: Replikation des geänderten Objekts, eventuell auch über weitere Sequencer



[MacIntyre & Feiner, Language-Level Support for Exploratory Programming of Distributed Virtual Environments]

Repo-3D: Architektur



- Repo-3D nutzt COTERIE zur Replikation eines Szenengraphen
- Aber: nicht gesamter Szenengraph wird repliziert, sog. *local variations* (z.B. Kameraposition) sind ausgenommen

COTERIE/ Repo-3D: Diskussion

- System war seiner Zeit weit voraus
- Fokus auf Interaktionstechniken, Rapid Prototyping
- Großer Overhead für Replikation
- Skaliert nur sehr bedingt

Tinmith

- Forschungsplattform, University of South Australia, W. Piekarski
- Fokus auf mobile outdoor AR
- Vermutlich effizientestes AR-System (läuft flüssig mit 40Hz auf 700MHz Pentium III Laptop mit GeForce 2GO)
- Objektorientierte Datenflussarchitektur

Tinmith: Architektur

- Datenflussarchitektur
- Kommunikation zwischen Komponenten über zentralen Object Store
  - Zugriff über Pfade (wie Files), z.B. `/human/body/camera`
  - Serialisierung im XML-Format möglich
  - Durch Serialisierung auch RMI-Mechanismus möglich
- Verteilung über handgestricktes P2P-Netz
- Wichtigste Komponente: Szenengraph

Tinmith: Hardwaresystem

- Allgemeines Problem: Wie bringt man viele verschiedene (experimentelle) Hardwaregeräte in einem mobilen Setup unter?
- Teilprobleme:
  - Stromversorgung
  - Interfaces (USB, seriell, Firewire)
  - Trackinggeräte: wo anbringen?
- Tinmith-Lösung: integrierter Rucksack

DWARF

- Distributed Wearable Augmented Reality Framework
- TU München (AG Prof. Gudrun Klinker), 2001 – heute
- 5 Doktoranden, ca. 30 Diplomanden
- Grundidee: löse Probleme in AR mit Methoden aus dem Software Engineering
- Fokus auf Softwarearchitektur, weniger auf konkreten Anwendungen

DWARF: Design Goals

- Verteilt:
  → flexible, modulare und wieder verwendbare Komponenten als Basis
- Fokus auf AR-Anwendungen
  → effizientes Laufzeitverhalten
- Soll aber auch kleine tragbare Rechner in Ubiquitous Computing Szenarien unterstützen
  → Peer-to-Peer Architektur

→ Ziel: Bringe AR-Interfaces in Ubicomp-Umgebungen

DWARF: Middleware

- Kernideen:
  - Komponenten (*DWARF Services*) als Bausteine
  - Datenflussarchitektur zur Laufzeit
  - Services haben *Needs* und *Abilities*
  - N&A werden durch *Service Manager* verschalten
  - Kommunikation in zwei Stufen:
    1. Verbindung über dezentralen Service Manager
    2. Laufzeitkommunikation direkt zwischen Peers
- Basistechniken:
  - CORBA zur Kommunikation mit dem Service Manager
  - OpenSLP zur Dienstfindung
  - CORBA (RMI), CORBA Notification Service (Events) und Shared Memory als P2P Kommunikationsprotokolle

DWARF: Tracking

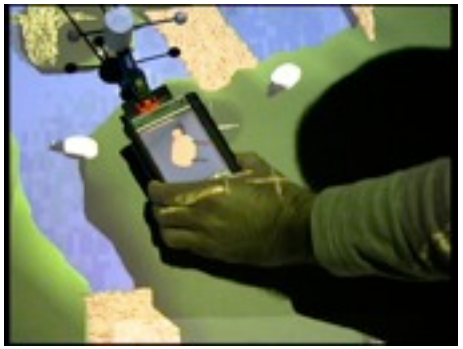
- DWARF Services kapseln COTS (*Custom Off The Shelf*) Hard- und Software (Intersense, ART, GPS NMEA, XSens, AR Toolkit)
- Gemeinsames Datenformat und Koordinatensysteme für alle Trackingdaten
- Verteilungsfeatures der DWARF-Middleware erlauben automatischen Auf- und Umbau komplexer Sensornetze zum Tracking (basierend auf formalem Modell, *Ubiquitous Tracking*)

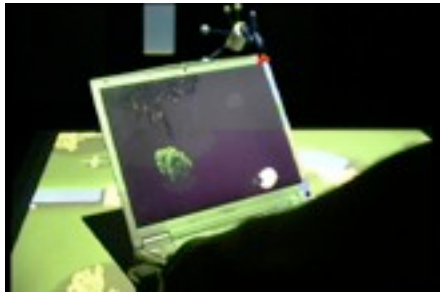
DWARF: User Interface

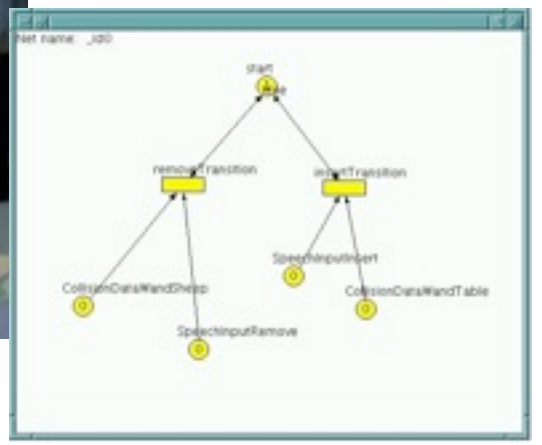
- User Interface Controller (UIC):
  - Kernidee: Taskflow
  - Modellierung durch Petrinetze
- Fokus auf Verwendung mehrerer Displays: Projektion, getrackte Displays (z.B. Tablet PC), HMD
- 3D-Viewer bietet abstrakte Schnittstelle zur Szenengraphmanipulation, auch Import von VRML/Open Inventor
- Ziel: Programmierung von AR-Interfaces durch Nicht-Experten im AR-Raum

DWARF: Beispiel SHEEP

- Shared Environment Entertainment Platform
- „Sinnfreies“ Schaferücken, mit mehreren Interaktionsgeräten und Displays
- Sehr verteilte Implementierung







[MacWilliams et al., Herding Sheep: Live System Development for Distributed Augmented Reality]

DWARF: Lessons Learned

- Verteilte AR ist möglich (auch wenn die derzeitige Implementierung besser sein könnte)
- „Echte“ Anwendungen brauchen hohe Flexibilität von DWARF nicht
  - Fokus auf Rapid Prototyping
  - Übernahme Techniken in Endprodukt
- Modellierung eines verteilten Systemzustands ist komplex
  - Zum Glück kommt sowas kaum in Forschungsprototypen vor…

DWARF: Lessons Learned (2)

- Koordination mehrerer Forscher auf einer Softwarebasis ist sehr schwierig
- Zwingende Voraussetzung: klar definiertes gemeinsames Ziel aller Beteiligten
  - Hier: baue ein verteiltes AR-Framework
  - Hier nicht: auf welche Systemteile fokussieren wir uns?
- Zu viel Freiheit im Systemdesign überfordert die meisten Entwickler
  - Templates als Entwurfsvorlagen
  - Einschränkung der Freiheitsgrade