Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Konzepte und Methoden zur Integration und Kommunikation in komplexen Systemen

Martin Konarski

Seminar “Prozesssteuerung und Robotik”

2009-01-28

Gliederung

- Einführung
- Design Pattern
- Simplex Architektur
- Real-Time Middleware
- Autonome Agenten

Einführung
Was ist ein DRE System?

- verteilte Hard- und Softwarekomponenten, die (meist) über Nachrichten kommunizieren
- können hochgradig heterogen sein
- Einschränkungen bezüglich
  - verfügbaren Resourcen
  - Quality of Service

Einführung
Resultierende Probleme

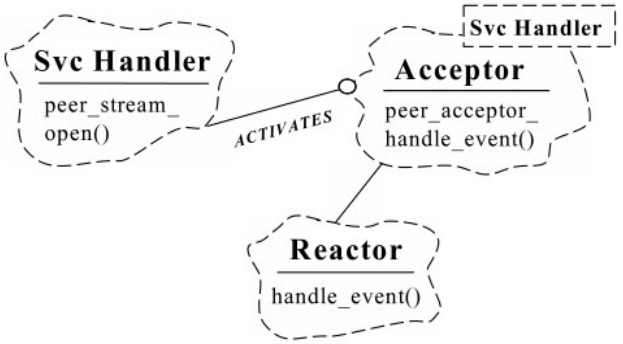
- viele heterogene Komponenten bedeuten hohen Entwicklungsaufwand und Kosten
- hardwarenahe Entwicklung ist fehleranfällig
- Plattformverfügbarkeit kann Entwicklung erschweren
- Systeme schwer zu aktualisieren oder an veränderte Bedingungen anpassbar
- einzelne Komponenten können kaputt gehen

Design Pattern

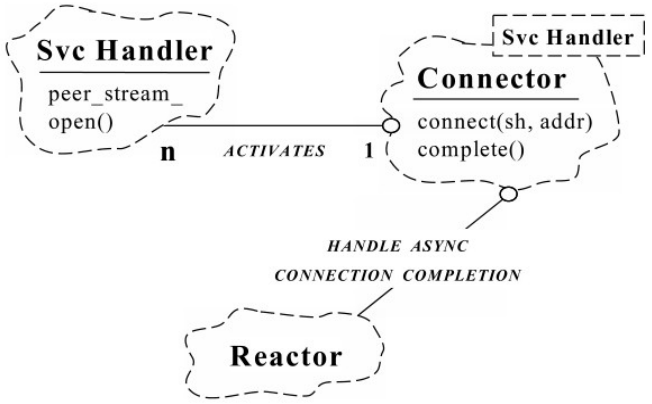
- Motivation: bewährte Lösungen für häufig auftretende Probleme zur Verfügung stellen
- sollen
  - Wiederverwendbarkeit
  - Übertragbarkeit
  - Erweiterbarkeit

  mit wenig Aufwand ermöglichen
- Bsp: Entkopplung von Verbindungsaufbau und anwendungsspezifischen Aufgaben mittels Acceptor, Connector und Reactor Pattern

Design Pattern
Acceptor und Connector



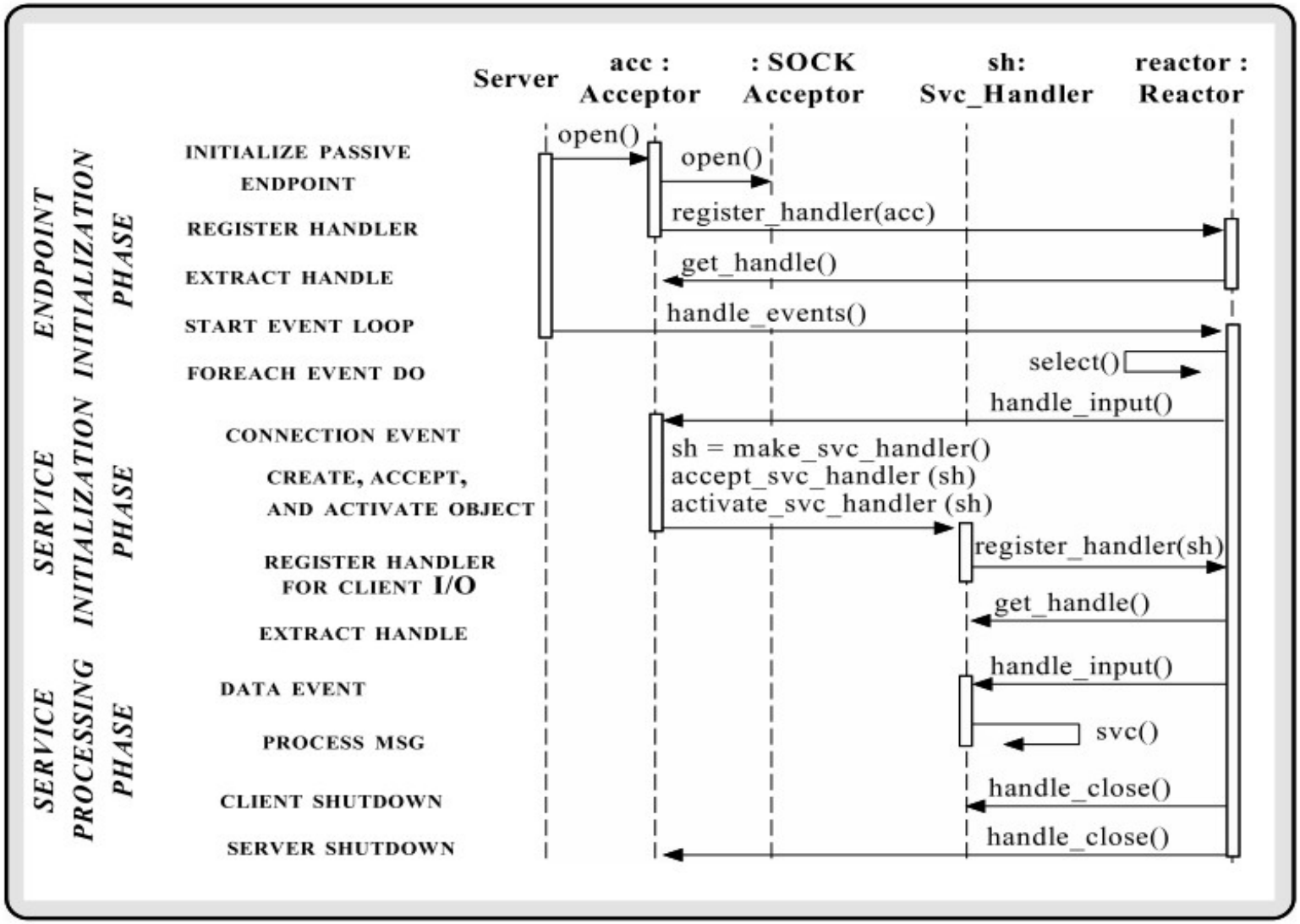
[2]



[3]

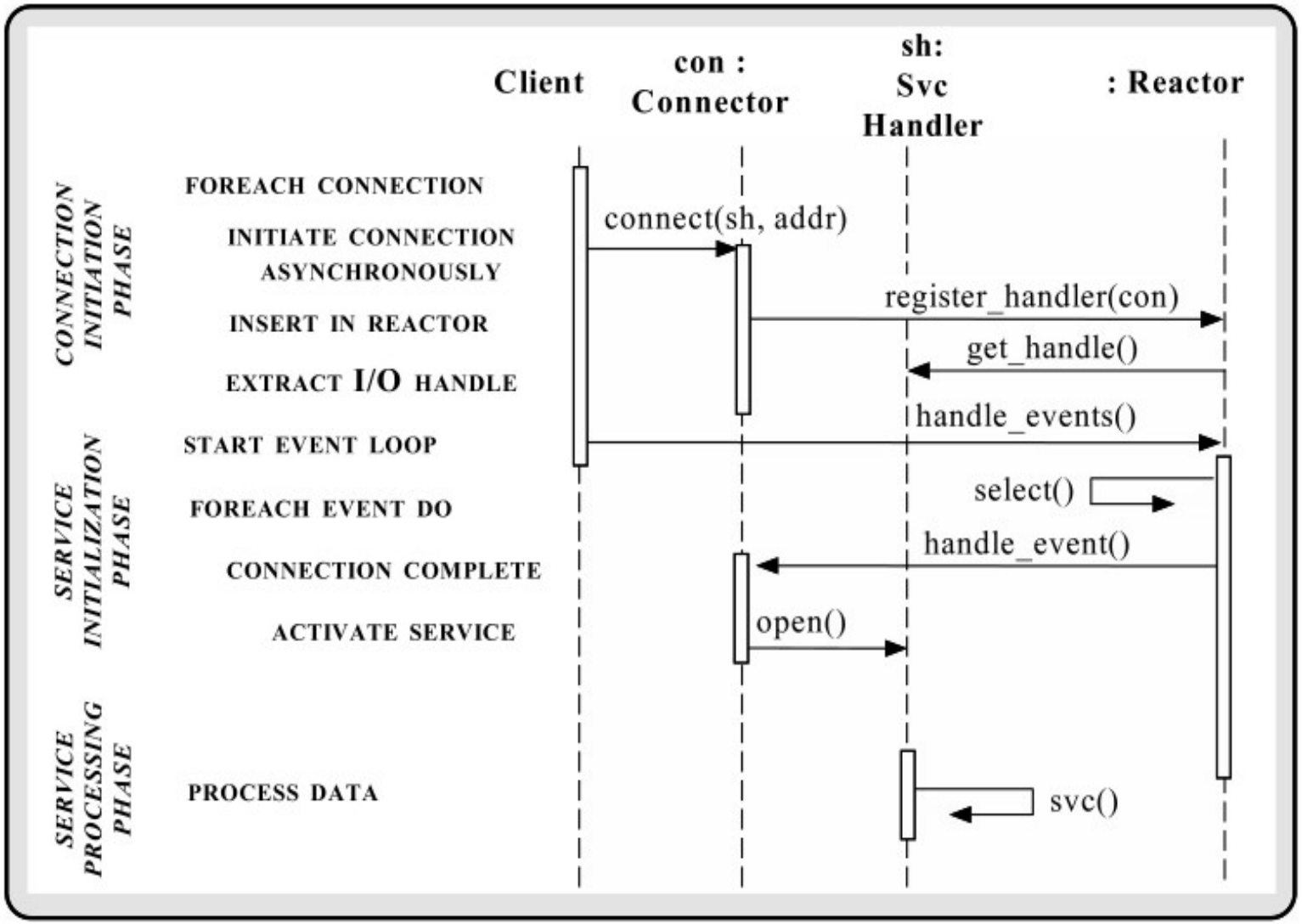
- Acceptor Objekt auf Server wartet auf Verbindungen
- Connector verbindet sich
- Reactor wickelt die Verbindung ab
- Service Handler von Acceptor und Connector können kommunizieren

Design Pattern
Acceptor Sequence Diagram



[2]

Design Pattern
Connector Sequence Diagram



[3]

Simplex Architektur

- entwickelt, um sichere Aktualisierungen von industriellen Computersystemen zu ermöglichen
- Downtime des Systems u.U. zu teuer oder nicht möglich
  → Veränderung des Systems während des Betriebs
- bietet verschiedene Möglichkeiten der Fehlertoleranz

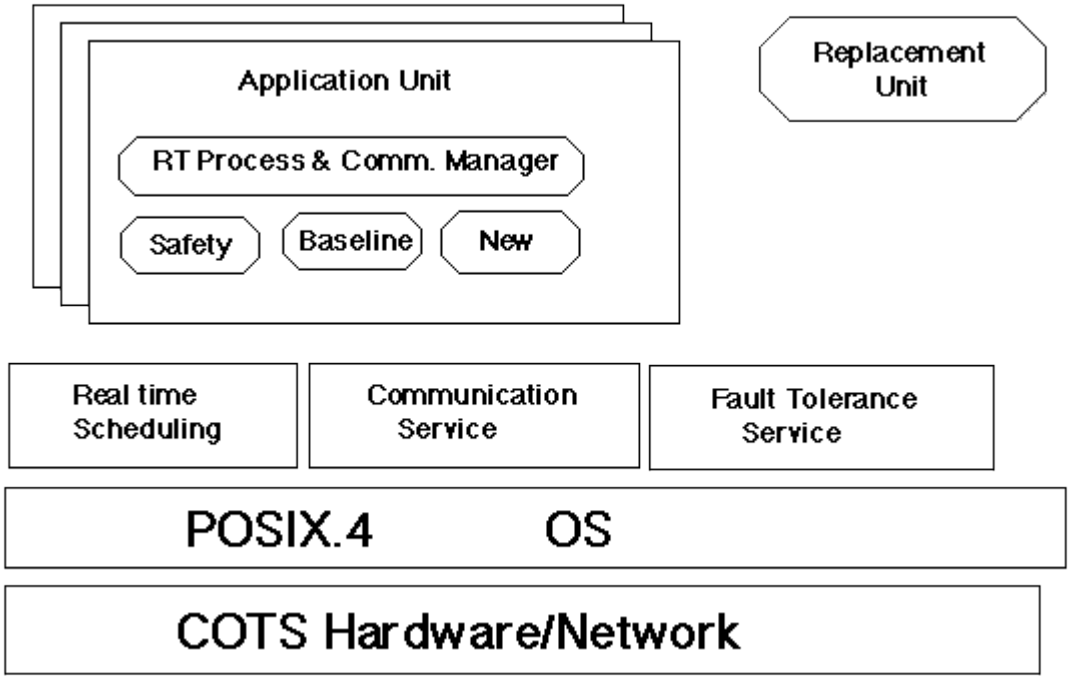
Simplex Architektur
Aufbau

- Replacement Units
  - Basiskomponenten der Simplex Architektur, alles ist Replacement Unit
  - bestehen aus min. einem Prozess und einem Communication Template
  - können durch Upgrade Transactions dem System hinzugefügt, entfernt, zusammmengeführt oder geteilt werden
- Sub-System Module
  - stellen autonome Subsysteme dar
  - können auf “logically interchangeable resources” ausgeführt werden
  - ermöglichen Implementation von Mechanismen um Softwarefehler zu tolerieren

Simplex Architektur
Aufbau

- Processor Configuration Manager (PCM)
  - verwalten Sub-System Modules
  - existiert einmal pro CPU
- System Configuration Manager (SCM)
  - verwaltet PCMs
  - erlaubt die Konfiguration des Systems (Topologie, Prozessoren, Netzwerke, “interchangable resources“, Variablenüberwachung, Toleranzen)

Simplex Architektur
Aufbau



[10]

- Sub-System Modules bestehen aus
  - Application Unit
  - Module Management Unit (Process Management, Upgrade Operation, Softwarefehler Behandlung)
  - optionaler Safety Unit (stellt sicher, dass Unit immer in sicheren Zustand versetzt werden kann)

Simplex Architektur
Upgrade Transactions

- Replacement Unit ersetzt analytisch redundante Replacement Unit

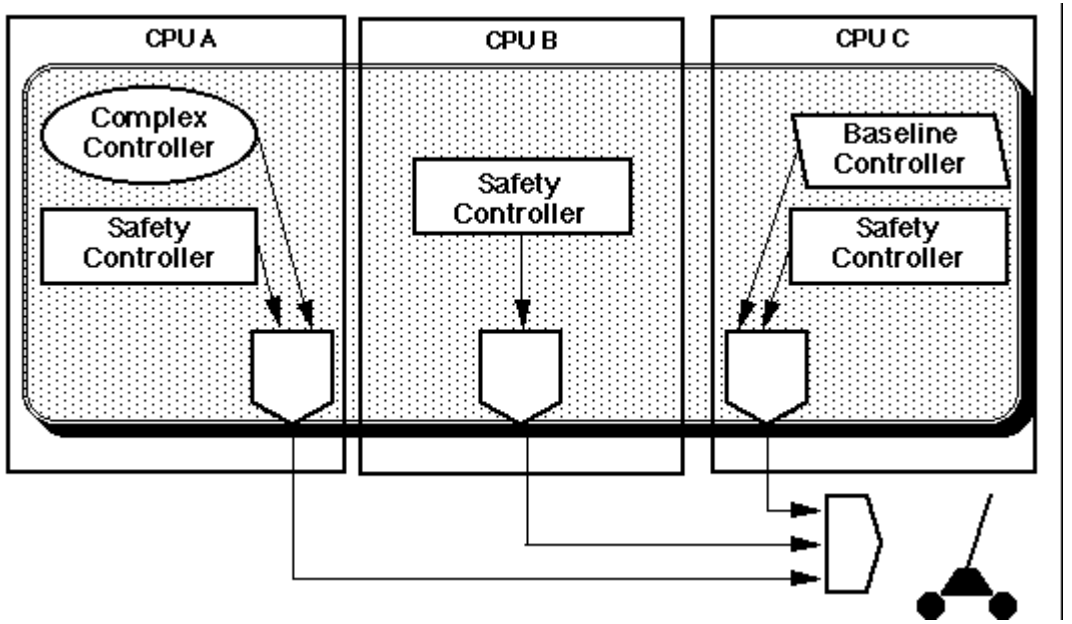
  1. neue Replacement Unit (NRU) erstellen

  2. Eingaben und Statusinformationen werden an NRU geliefert, Ausgabe wird überwacht, jedoch nicht benutzt

  3. Ausgabe der NRU ist in akzeptablem Rahmen

  4. Ausgabe der alten Replacement Unit abeschalten, Ausgabe der NRU einschalten und die alte Unit löschen

Simplex Architektur
Fehlertoleranz



[10]

- Toleriert werden kann
  - beliebiger Fehler in einem der Systeme
  - Absturz eines zweiten Systems
  - Fehler in den nicht-Safety Controllern des dritten Systems

Simplex Architektur
Einsatzgebiete

- generell sicherheitskritische Systeme
- F-35 Joint Strike Fighter
- INSERT Auto-Pilot System

Real-Time Middleware

- Middleware bietet viele Vorteile für DRE Systeme, unter anderem:
  - Abstraktion von fehleranfälligen APIs
  - Heterogenität der einzelnen Komponenten wird verborgen
  - location, cuncurrency, mobility, replication und failure transparency

- Standard Middleware ist jedoch ungeeignet, weil:
  - keine Prioritäten
  - keine Zeitbedingungen / -schranken
  - kein Wissen über WCET von Funktionsaufrufen und Event-Behandlung
  - keine Echtzeitscheduling-Mechanismen

Real-Time Middleware
Real-Time CORBA

- Erweiterung des bewährten CORBA Standards

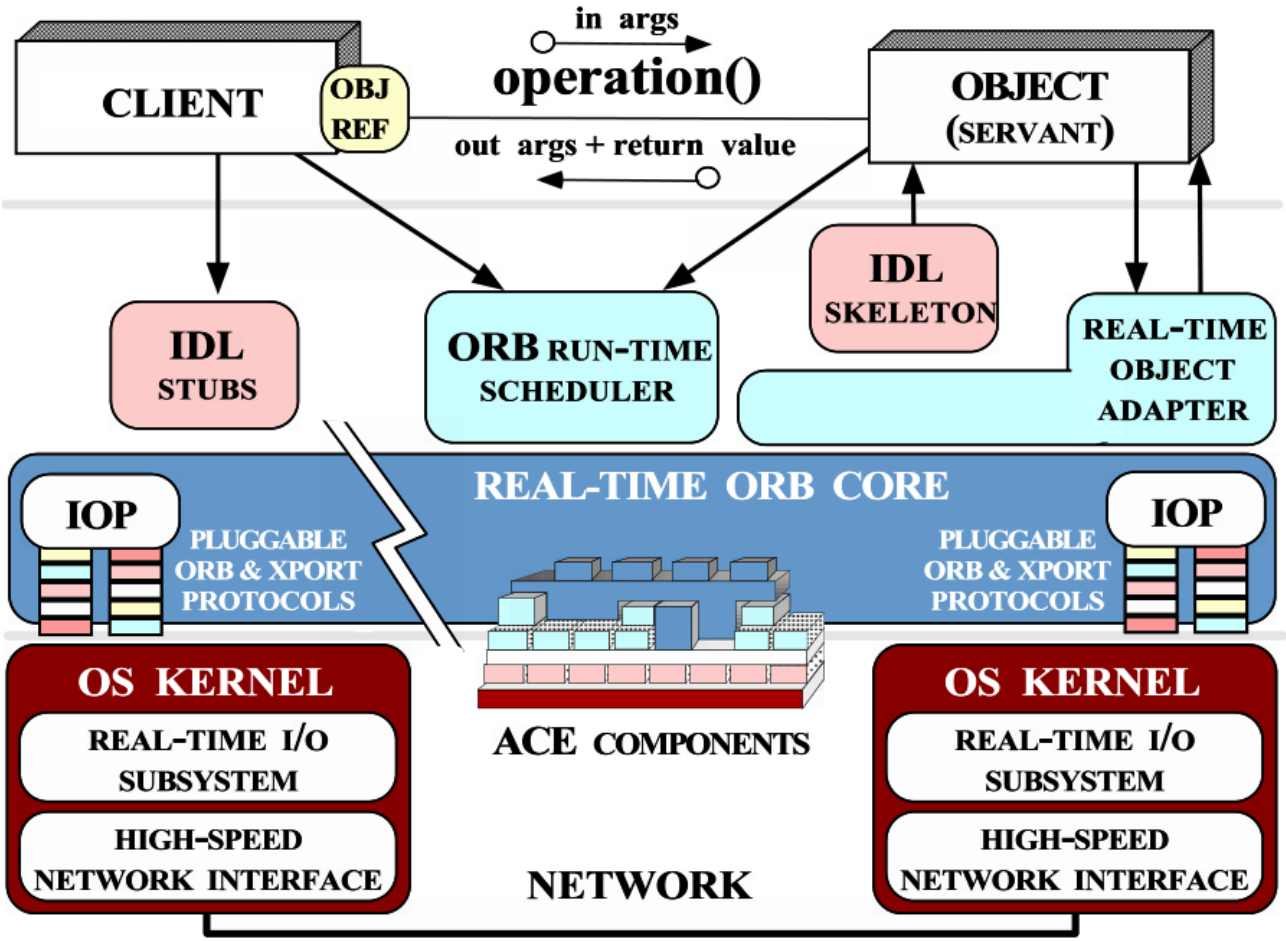
  - Processor Resource Management: Operationen der CORBA-Objekte werden globale Echtzeitprioritäten zugewiesen
  -Memory Management: Anzahl der Threads, deren Speichergröße und Queuegröße für CORBA-Requests kann festgelegt werden
  - Network Resource Management: Auswahl und Konfiguration der verfügbaren Netzwerkprotokolle, Clients können private Verbindung zum Server aufbauen oder mehrere Verbindungen zum Server nutzen

Real-Time Middleware
Real-Time CORBA

- RT CORBA für eingebette Systeme zu groß
  → Minimum CORBA Spezifikation
  Subset des Standards speziell für eingebettete Systeme

- Implementationen zum Beispiel „TAO – The ACE ORB“ und „RTZen”

Real-Time Middleware
TAO

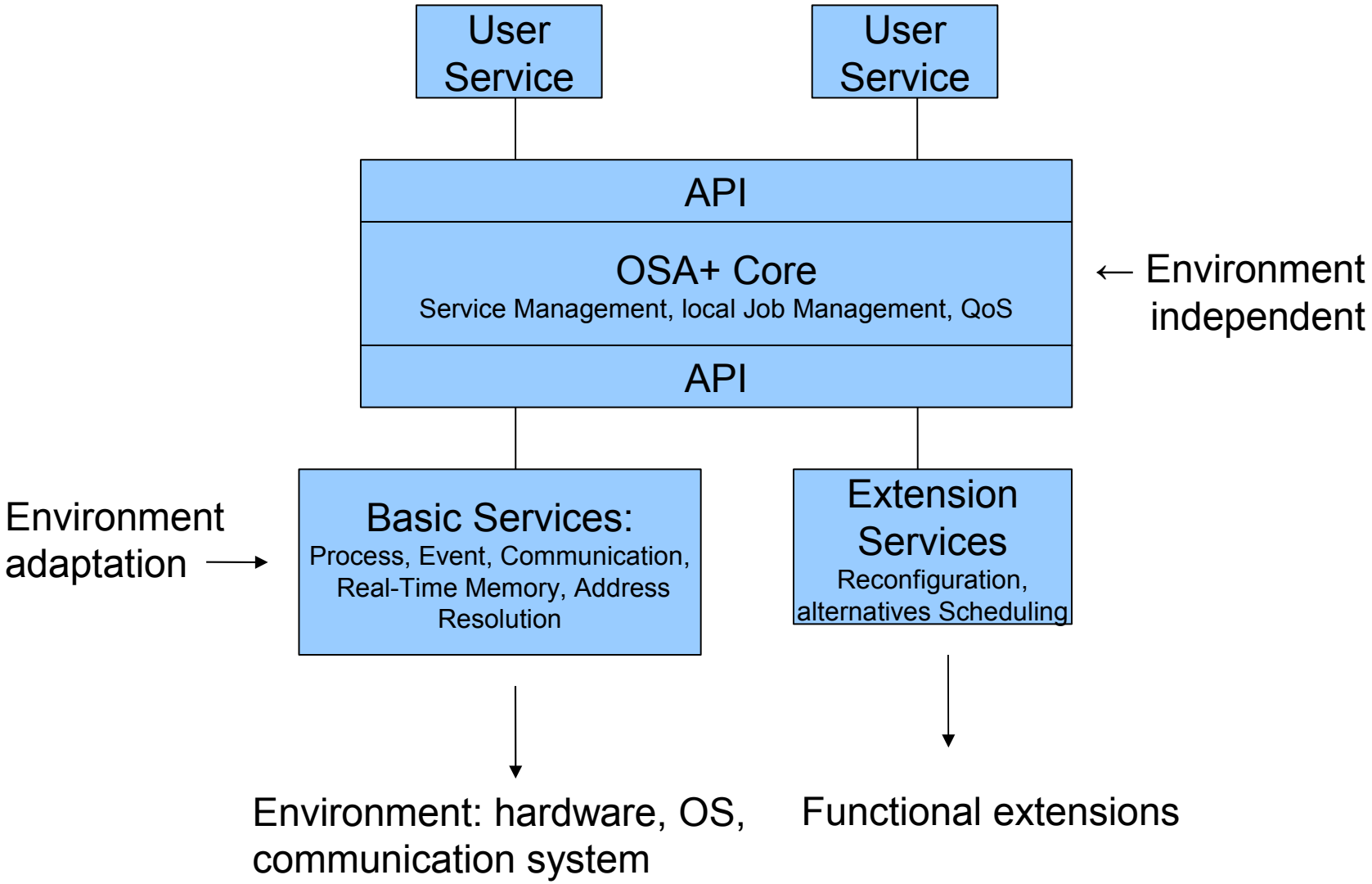


[8]

Real-Time Middleware
OSA+

- auf Microkernel-Ansatz basierende Service orientierte Middleware
- speziell für DRE Systeme entwickelt
- Services werden im OSA+ Core registriert und kommunizieren über Jobs
- Jobs bestehen aus Order/Result und haben eine Priorität
- Services übernehmen Priorität ihres aktuellen Jobs

Real-Time Middleware
OSA+ Aufbau



nach [5]

Autonome Agenten
Definition

[..] an agent is a computer system situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives. Autonomy [is used] in the sense that the system should be able to act without the direct intervention of humans (or other agents) [..].

[9]

Autonome Agenten
Beispiele

- Prozesssteuerung
  Process Controller nimmt Umgebung (Sensordaten und andere Eingaben) wahr und agiert entsprechend seines Ziel selbständig
- Aufklärungsroboter
- E-Mail Filter
- ...

Quellen

[1] Computers as Components, Principles of Embedded Computer System Design; W. Wolf; Elsevier Inc.; 2005

[2] Acceptor, A Design Pattern for Passively Initializing Network Services; D. C. Schmidt

[3] Connector, A Design Pattern for Actively Initializing Network Services; D. C. Schmidt

[4] Reactor, An Object Behavioral Pattern for Demultiplexing and Dispatching Handles for Synchronous Events; D. C. Schmidt

[5] Scalable and Efficient Middleware for Real-Time Embedded Systems. A Uniform Open Service Oriented, Microkernel Based Architecture; F. Picioroaga; PhD Thesis; 2004

[6] A Middleware Approach for Dynamic Real-Time Software Reconfiguration on Distributed Embedded Systems; E. Schneider; PhD Thesis; 2004

[7] A Software Architecture for Dependable and Evolvable Industrial Cumputing Systems; L. Sha, R. Rajkumar, M. Gagliardi; 1995

[8] Applying a Pattern Language to Develop Extensible ORB Middleware; D. C. Schmidt, C. Cleeland

[9] Applications of Intelligent Agents; N. R. Jennings, M. Wooldridge

[10] Simplex Architecture; http://www.sei.cmu.edu/simplex/index.html

[11] A Portable, Autonomous, Urban Reconaissance Robot; L. Matthies, Y. Xiong, R. Hogg, D. Zhu, A. Rankin, B. Kennedy, M. Hebert, R. Maclachlan, C. Won, T. Frost, G. Sukhatme, M. McHenry, S. Goldberg

[12] Kommunikation in Echtzeit Systemen; G. Färber; Seminarband; 2006

[13] Updating RT Embedded Software in the Field, L. Sha, 2002