

# **Organic (Autonomic) Computing for Embedded Real-time Systems**

Uwe Brinkschulte

Institute of Process Control, Automation and Robotics,  
University of Karlsruhe, Germany

Theo Ungerer

Institute of Computer Science,  
University of Augsburg, Germany

# Outline of the Presentation

## 1. Our Contributions to Embedded Real-time Systems

1.1 The Komodo Project

1.2 The Middleware OSA+

## 2. Perspectives for Organic (Autonomic) Computing in Real-time Systems

2.1 General

2.2 Helper-Threads, Multithreaded Hardware and Middleware

2.3 The Use of Control Theory

2.4 Dynamic Reconfiguration on Middleware-Level

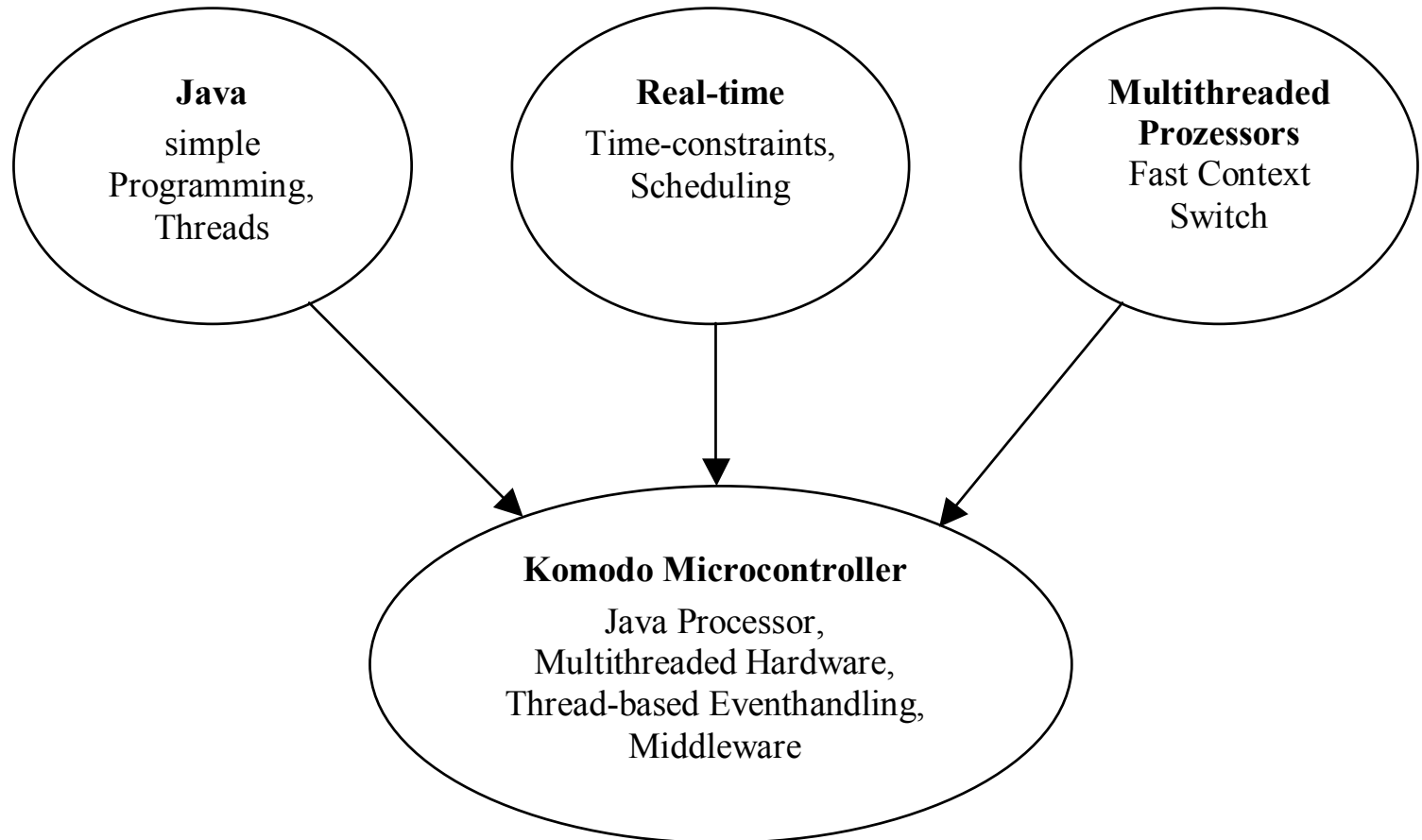
2.5 Possible Platforms

## 3. Conclusions

# 1. Our Contribution to Embedded Real-Time Systems

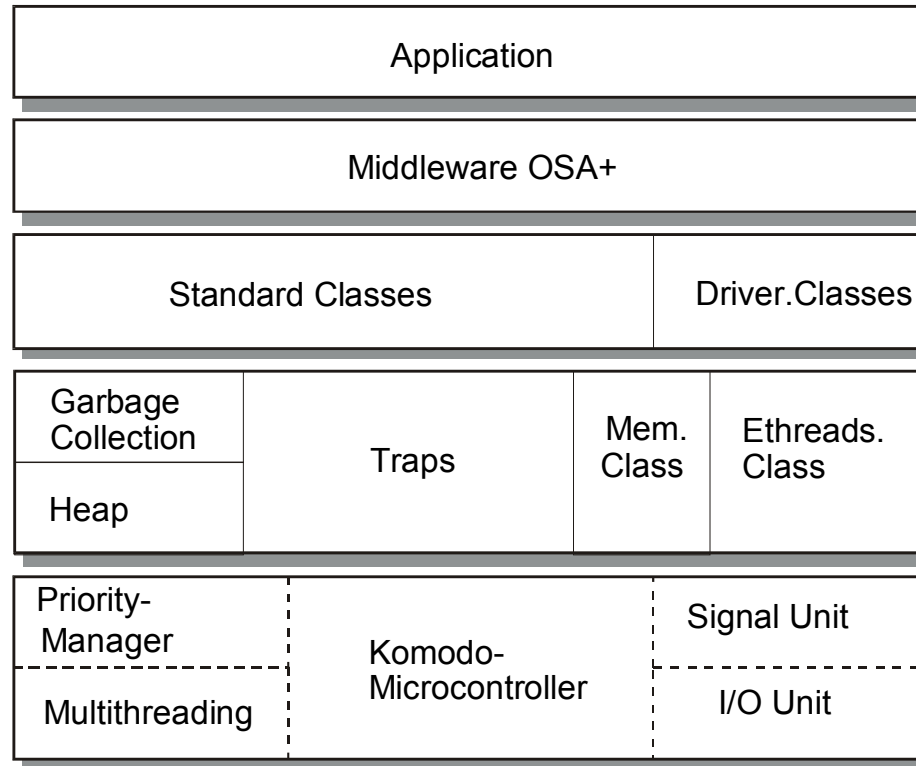
## 1.1 The Komodo Project

Basic ideas:



# 1. Our Contribution to Embedded Real-Time Systems

## The layers of Komodo



# 1. Our Contribution to Embedded Real-Time Systems

---

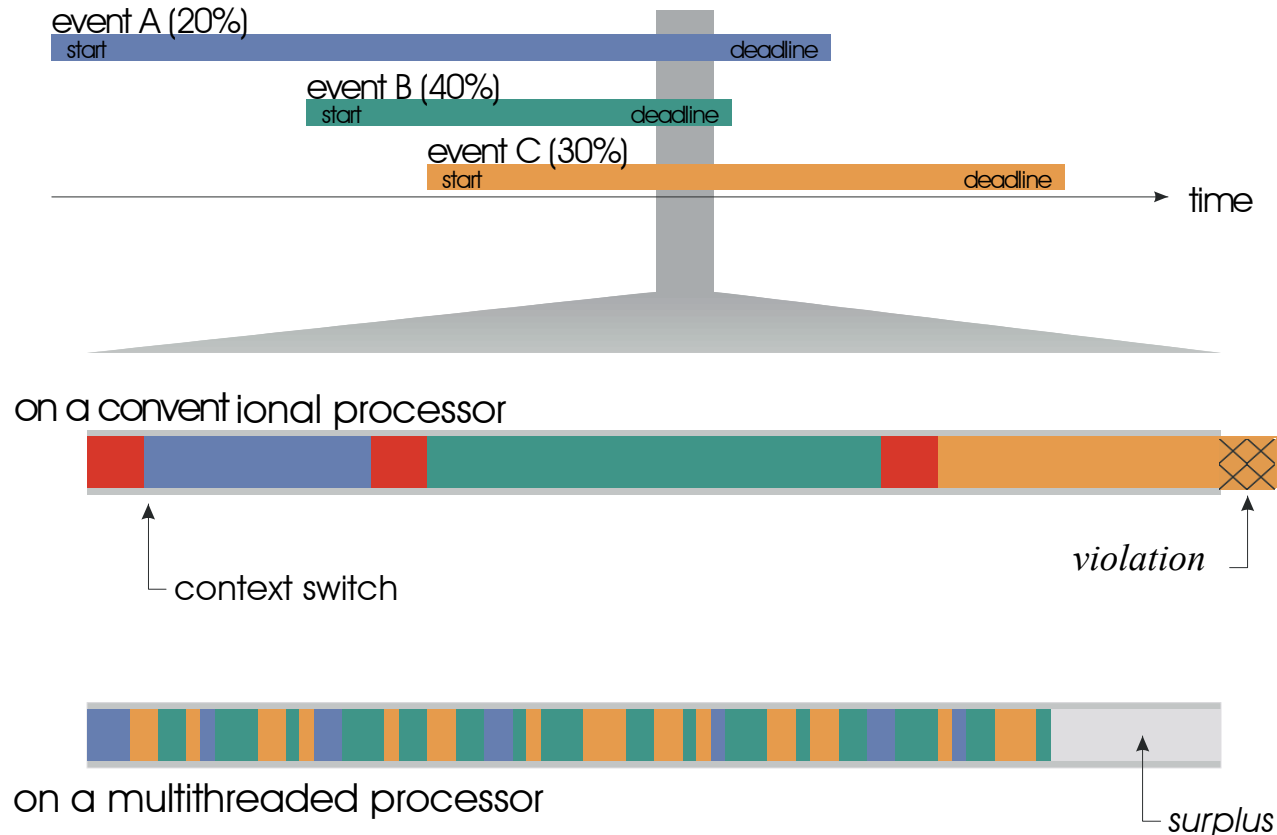
Real-time Scheduling in Hardware:

- Fixed Priority Preemptive (FPP)
- Earliest Deadline First (EDF)
- Least Laxity First (LLF)
- Guaranteed Percentage (GP)

The scheduling is executed in hardware to allow a scheduling decision within one clock cycle.

# 1. Our Contribution to Embedded Real-Time Systems

GP scheduling: assign each thread a specific percentage of the processor's performance over a short time period (100 processor cycles)



# 1. Our Contribution to Embedded Real-Time Systems

## Advantages of GP:

- strict timing isolation of threads
- guaranteed response times and data rates for multiple threads
- simple detection of overload conditions ( $>100\%$ )
- allows helper threads without changing the real-time behavior
- fine-grain realization on a multithreaded processor core

# 1. Our Contribution to Embedded Real-Time Systems

## **ISTs and Helper Threads:**

- **ISTs** (Interrupt service threads) instead of ISRs (Interrupt service routines)
  - ISTs run concurrently to the application in a separate thread slot
  - ISTs are an example for a more general concept:
- **Helper threads** perform OS tasks concurrently to the application in own thread slots
  - E.g. in Komodo: event handling by ISTs, garbage collection, dynamic class loading

# 1. Our Contribution to Embedded Real-Time Systems

## 1.2 The Middleware OSA+

lean and scalable middleware for embedded real-time systems

OSA+ :           **Open System Architecture**  
                      - **Platform for Universal Services**

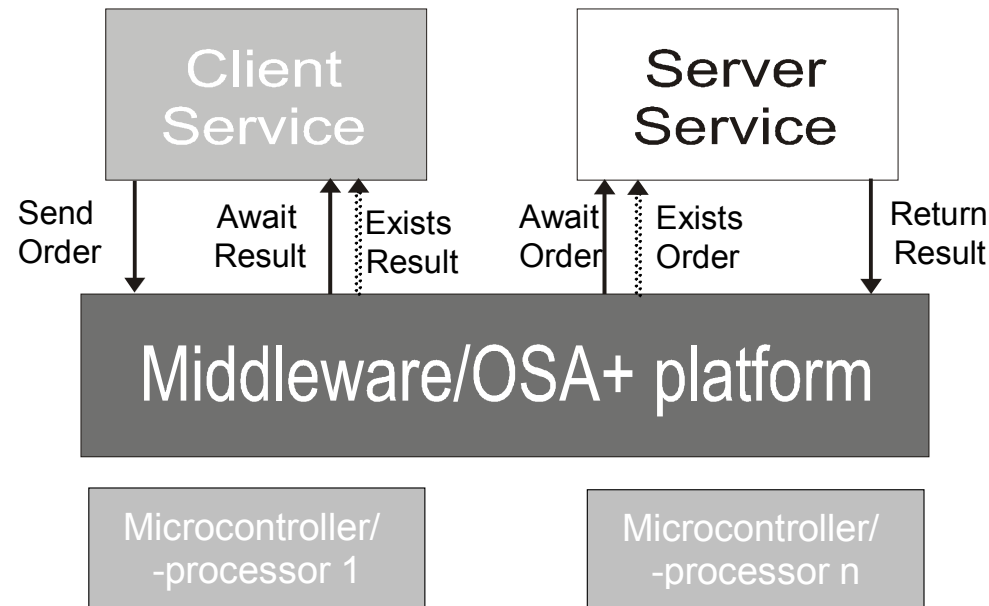
Goals:

- Real-time
- Scalability
- Efficiency
- Support of weak hardware (microcontrollers ...)

# 1. Our Contribution to Embedded Real-Time Systems

## Basic Concepts:

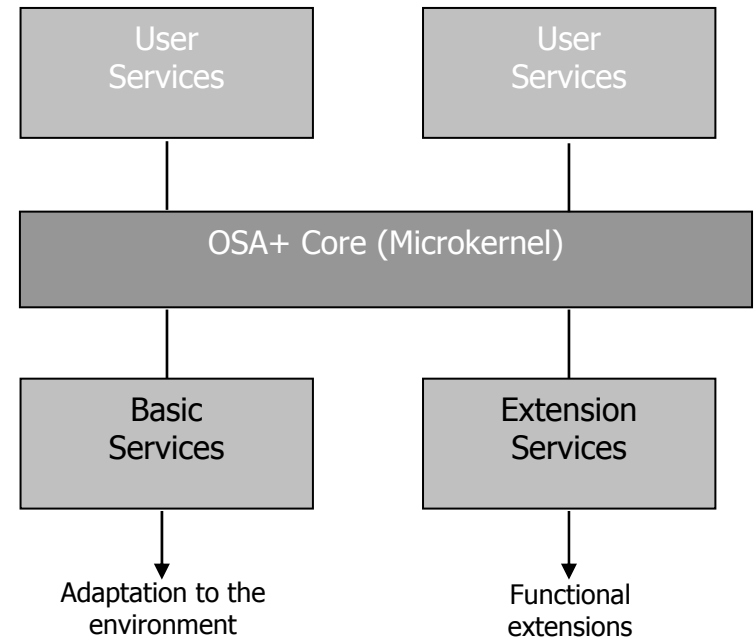
- Service oriented architecture (contr. to CORBA, DCOM, RMI)
- Communications done by jobs  
Job = Order + Result
- Synchronous Communications:
  - SendOrder, AwaitResult - AwaitOrder, ReturnResult
- Asynchronous Communication:
  - SendOrder, ExistResult - ExistOrder, ReturnResult



# 1. Our Contribution to Embedded Real-Time Systems

## Realization:

- The **micro-kernel concept** well known from RTOS is used:
  - **a very small core** with the minimal basic functionality
  - **services** as small building blocks to extent the basic functionality
- High scalability is a direct result of the micro-kernel concept



# 1. Our Contribution to Embedded Real-Time Systems

## ➤ Scaling by micro-kernel and services:

- Micro-kernel
  - local sequential services
- Micro-kernel + Process-service
  - local sequential and parallel services
- Micro-kernel + Process-service + Communication-service
  - local and distributed sequential and parallel services
- Micro-kernel + Event-service
  - local sequential, time-triggered services
- ...

# 2. Perspectives for Organic (Autonomic) Computing in Real-time Systems

## 2.1 General

The autonomic (organic) computing principles

- (self-organization)
- self-configuration
- self-optimization
- self-healing
- self-protection
- context awareness, (adaptive)

bring benefits to the design of embedded real-time systems.

Here are some ideas on how to bring these two fields together.

# 2. Perspectives for Organic (Autonomic) Computing in Real-time Systems

## 2.2 Helper-Threads, Multithreaded Hardware and Middleware

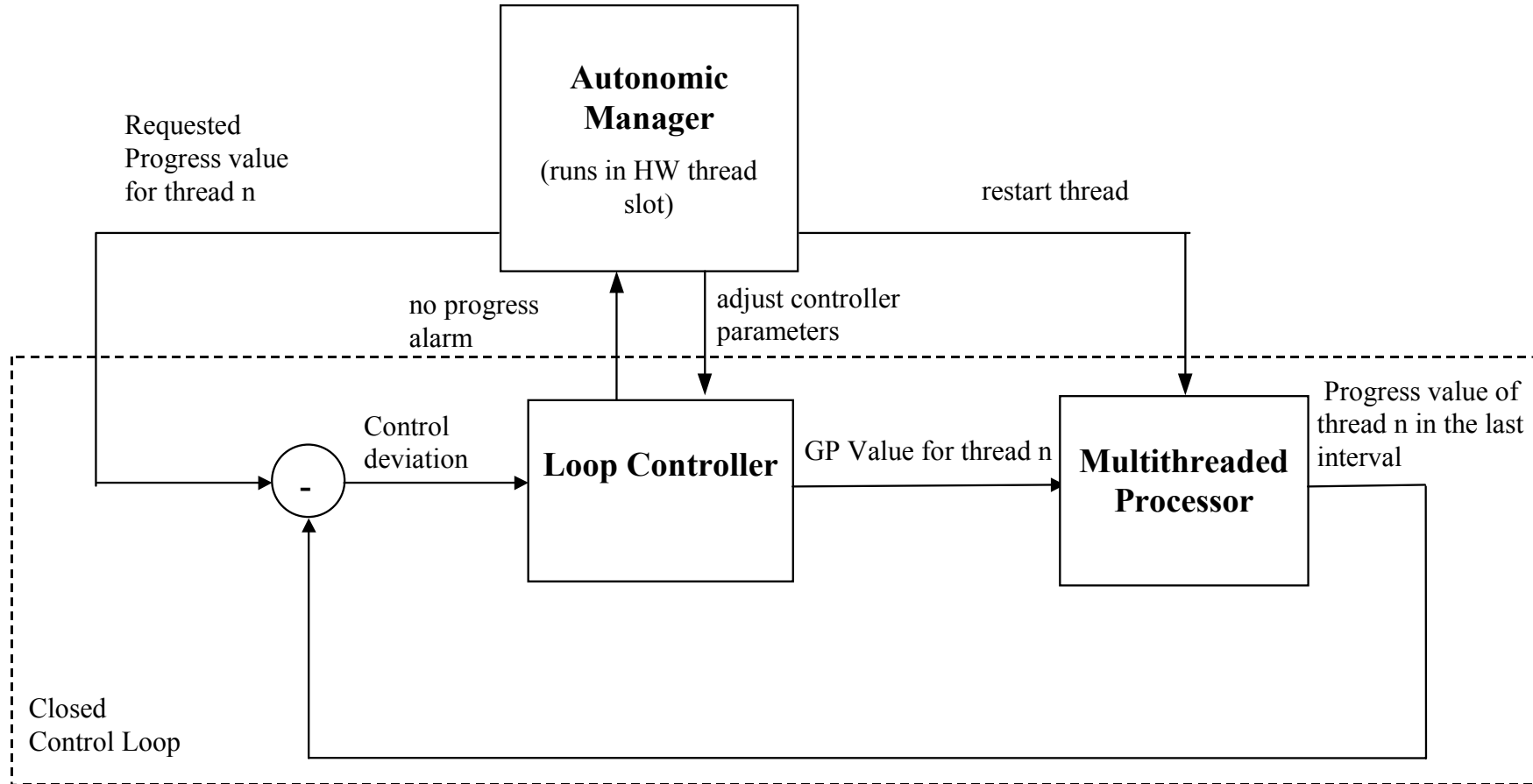
- The helper thread-concept can be used to build an **Autonomic Manager** as **general organic (autonomic) supervisor**
- This manager **monitors** the system and **adapts** it according to the current needs and **organic (autonomic) principles**
- It can run in a **thread-slot** of a multithreaded processor without affecting the **real-time behavior** of other tasks (e.g. in combination with hardware GP scheduling)
- Transported on the **middleware-level** (helper-thread -> helper service) a **distributed system** can be monitored and adapted

# 2. Perspectives for Organic (Autonomic) Computing in Real-time Systems

## 2.3 The Use of Control Theory

- Control theory is a well known principle for the **self-configuration** and **self-optimization** of systems
- It can be used to evaluate and tweak the overall system behavior and even initiate **self-healing**
- Example: adapt the real-time scheduling parameters in GP scheduling:
  - threads give progress signals
  - closed control loop autonomously adapts the percentages to reach a preferred progress rate
  - missing progress signals indicate failure => watchdog
- The **autonomic manager** can adapt the control loop and initiate necessary actions (e.g. restart of a thread)

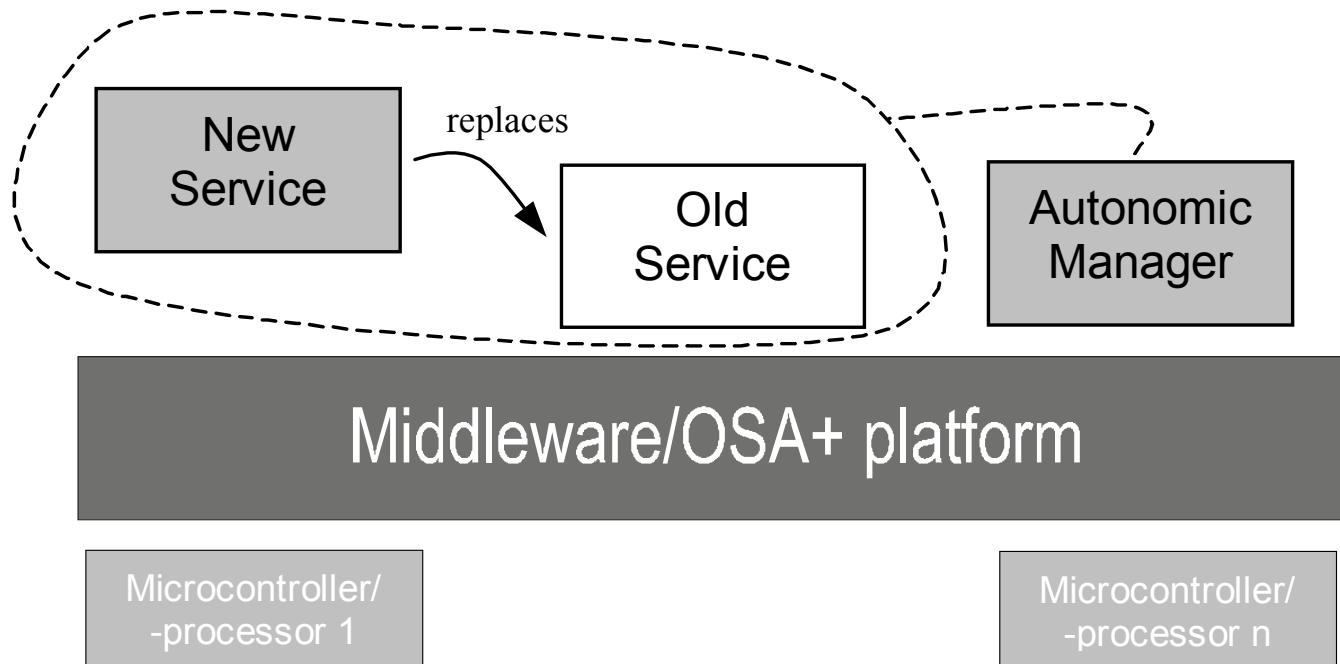
# 2. Perspectives for Organic (Autonomic) Computing in Real-time Systems



# 2. Perspectives for Organic (Autonomic) Computing in Real-time Systems

## 2.4 Dynamic Reconfiguration on Middleware-Level

The autonomic manager can initiate a reconfiguration on middleware-level by exchanging and replacing services to adapt, maintain or repair the system



## 2. Perspectives for Organic (Autonomic) Computing in Real-time Systems

### Challenges:

- Reconfiguration must be done while the system is running
- Reconfiguration must be done in real-time
- State must be transferred while the system is running and in real-time
- Trade-off between blackout time and reconfiguration time is necessary
- ...

# 2. Perspectives for Organic (Autonomous) Computing in Real-time Systems

## 2.5 Possible Platforms

Possible platforms to implement and evaluate our ideas:

- the Komodo Microcontroller with the Java environment
- an industrial multithreaded microprocessor (e.g. Pentium IV) with Linux OS

The Linux kernel 2.6 thread scheduler has to be adapted to run

+ application threads

+ autonomic manager

efficiently on the multithreaded microprocessor

# 3. Conclusions

- Contributions:
  - Komodo Project – multithreaded Komodo processor and real-time middleware OSA+
  - Guaranteed percentage scheduling, ITSs and helper threads
- Perspectives:
  - Investigate multithreaded processors as basis for organic (autonomic) computing principles
  - Investigate (real-time) operating system and middleware issues
  - Autonomic manager as helper thread supervises system due to organic (autonomic) principles
  - Use of control theory for self-organization, self-configuration, ...

# 3. Conclusions

- Roadmap:
  - The first two years
    - Evaluation of multithreaded hardware for organic real-time computing
    - Evaluation of control theory for organic real-time computing
    - Developing adaptive and reconfigurable middleware-structures
  - The third year
    - Prototypic implementation of a hardware/software environment for organic computing based on the results of the first two years
  - The fourth year
    - Selecting and implementing a demo application to evaluate the developed system
- Possible partners from industries:
  - Infineon
  - Kontron

# Technical Appendix

## Processor Core

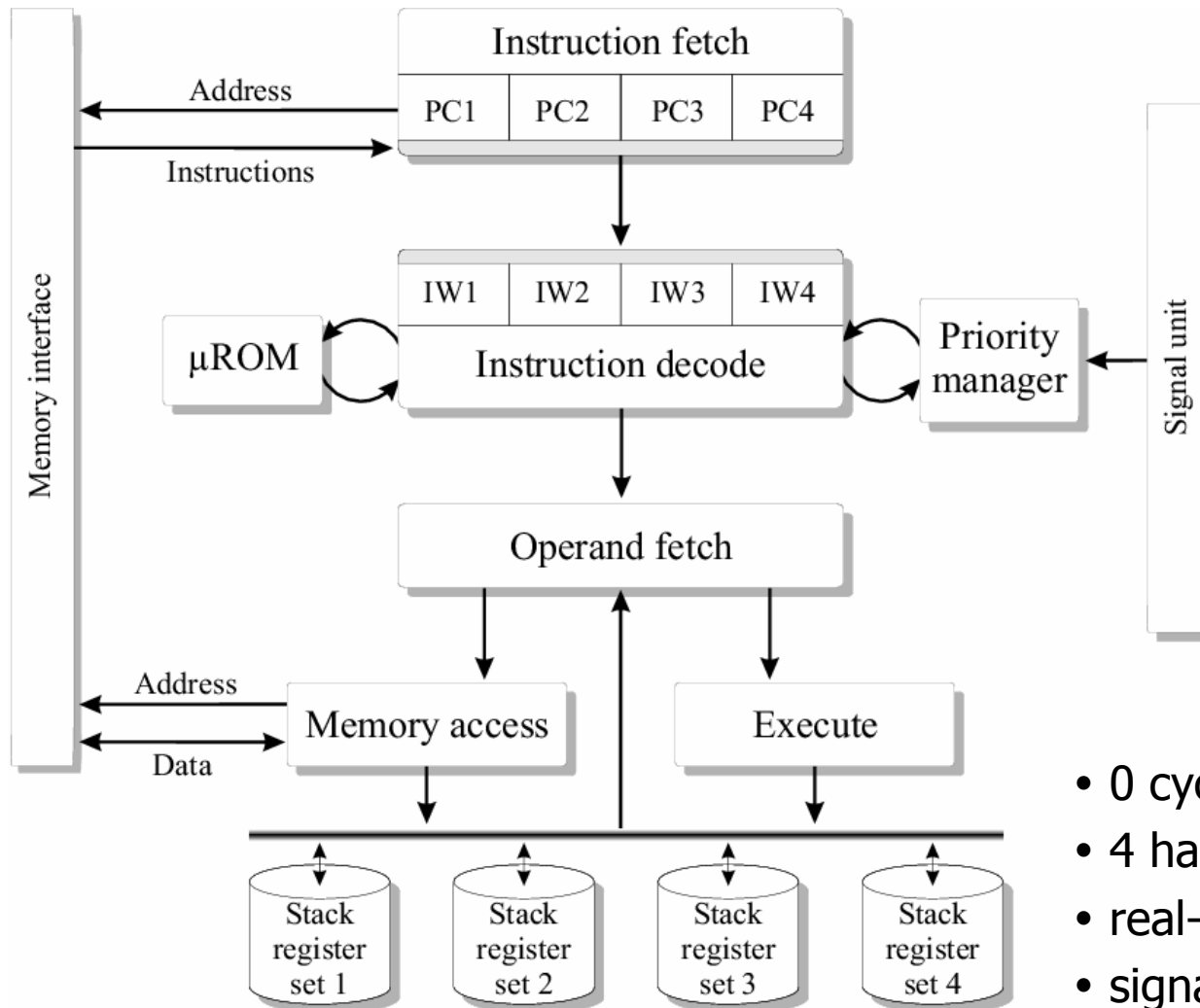
### Java Processor

- direct execution of Java bytecode
- stack register sets
- support for garbage collection

### Multithreaded Processor

- executes more than one thread at the same time in the pipeline
- contains several program counters and register sets
- ↓ extremely fast context switches
- ↓ latency bridging
- ↓ efficient real-time event handling

# Technical Appendix



- 0 cycle context switch
- 4 hardware threads
- real-time scheduling in HW
- signal unit for external events

# Technical Appendix

## GP Classes:

- *Exact*: a thread gets **exactly** the requested percentage
- *Minimum*: a thread gets **at least** the requested percentage
- *Maximum*: a thread gets **at most** the requested percentage
- *Non real-time*: a thread gets **all the rest**

The workload of *exact* and *minimum* threads must not exceed 100%

The latency usage gain above 100% can be used by *minimum* and *non real-time* threads

# Technical Appendix

An OSA+ job is used for:

- Communication

  - by exchanging orders and results

- Synchronisation

  - by the sequence of orders

- Parallelism

  - by parallel orders

- Real-time operations

  - by timing constraints given in the order

# Technical Appendix

## ➤ Functions/properties of the OSA+ micro-kernel:

- contains local service management
- supports sequential service execution
- allows local communication through jobs
- realizes the user-interface
- is completely independent from the underlying operating- and communication system

=> This functionality is used in 100% of all applications

# Technical Appendix

## ➤ Service classes:

- Basic Services:
  - Process-Service: service scheduling, synchronisation
  - Memory-Service: dynamic memory management
  - Event-Service: real-time events and time-based job execution
  - Communication-Service: non-local job delivery
  - Address-Service – Localization of non-local services
- Extension Services: Error-logging, cryptography, helper services, ...
- User Services