

FZI Forschungszentrum Informatik

an der Universität Karlsruhe



Forschungsbereich
Systementwurf in der Mikroelektronik
Prof. Dr. W. Rosenstiel

Der Einsatz von Jini für die Realisierung durchgängiger Steuerungskonzepte in verteilten eingebetteten Systemen

Stephen Schmitt

Gliederung

1 Motivation
- Anforderungen an eingebettete Systeme
- Probleme und Lösungsmöglichkeiten

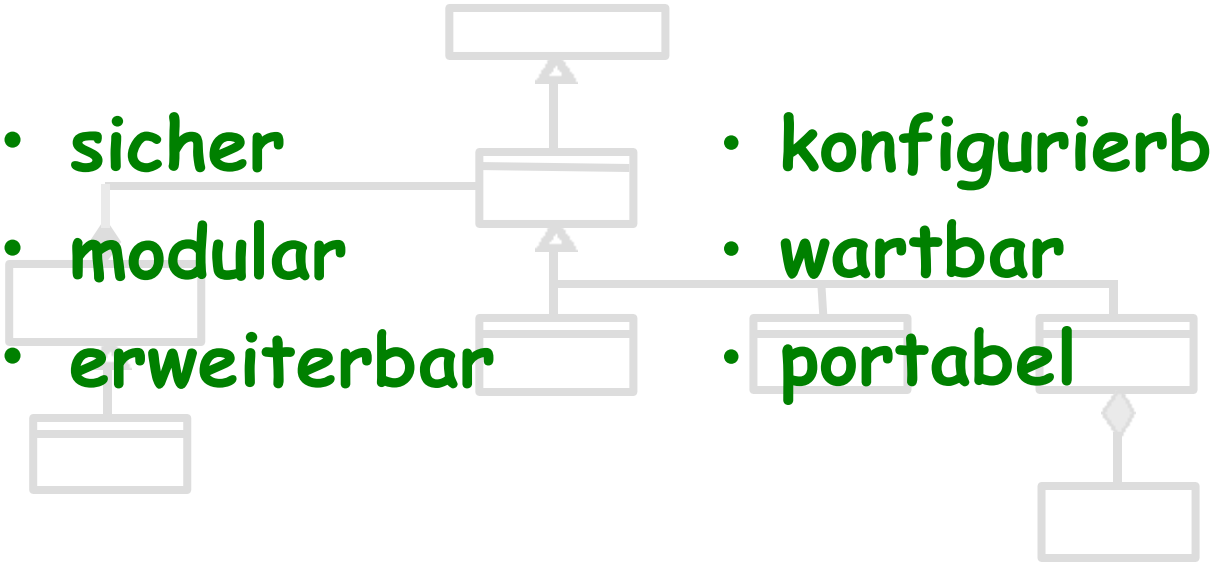
2 Jini-Technologie
- Überblick
- Verwendete Features

3 Jini in der industriellen Automation
- Architekturen von Jini-Anwendungen
- HW/SW-Architektur des Systems

4 Ergebnisse
- Vorteile durch Jini
- Probleme

5 Zusammenfassung und Ausblick

Anforderungen an eingebettete Systeme



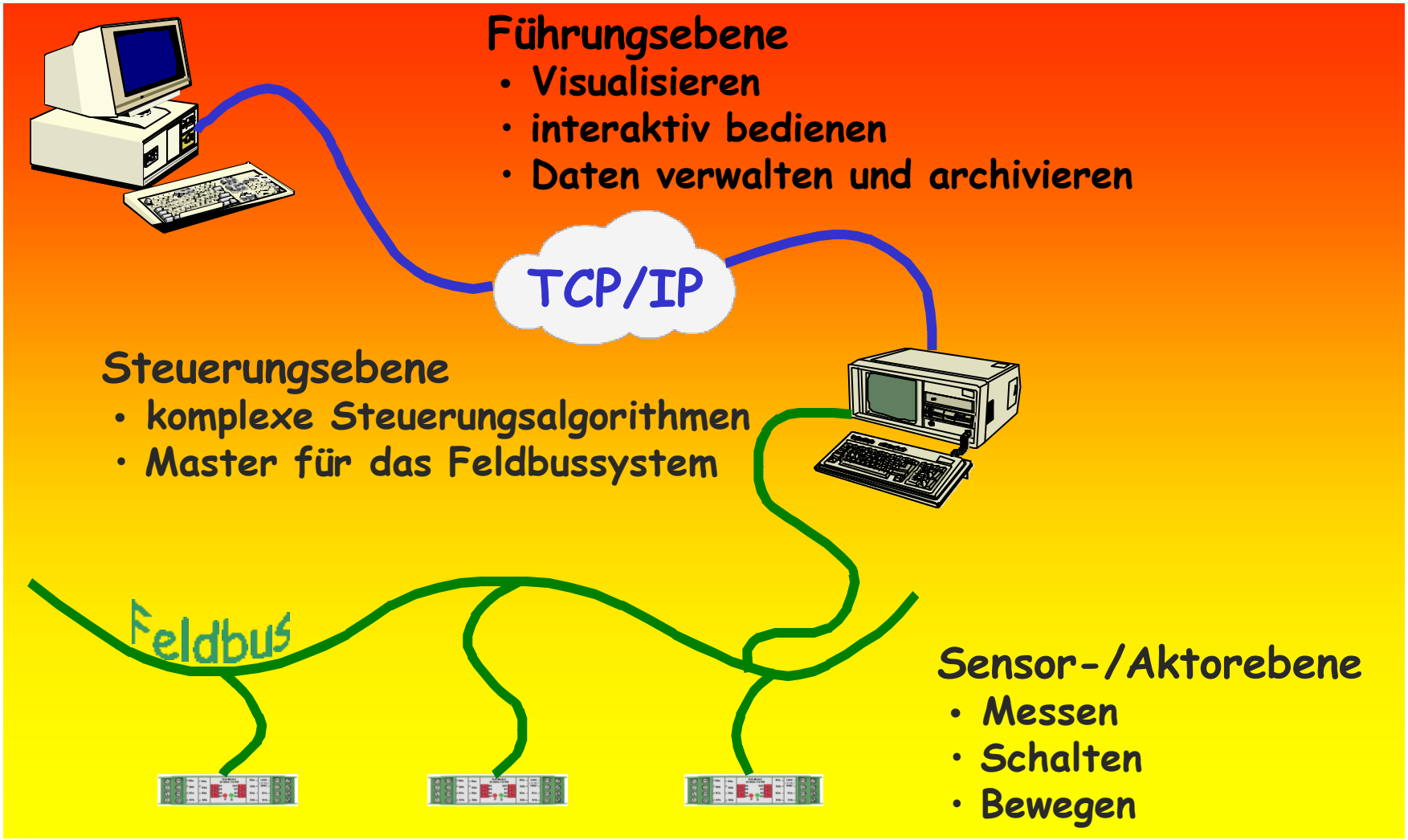
Software:

- sicher
- modular
- erweiterbar
- konfigurierbar
- wartbar
- portabel

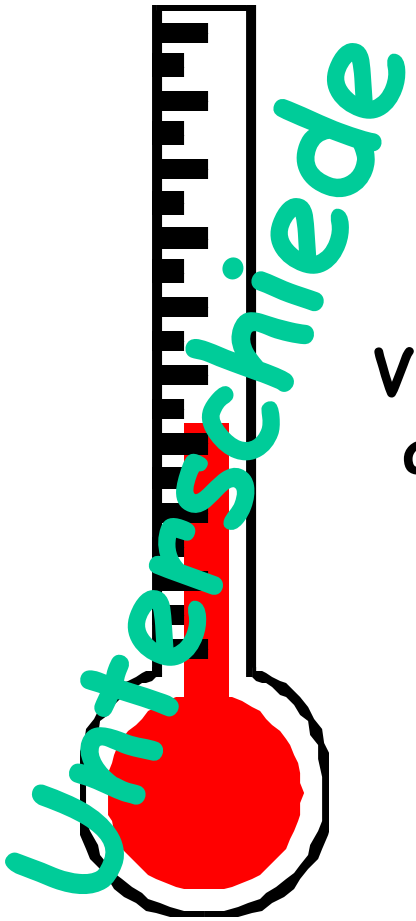
Hardware:

- Leistungsfähige Mikrocontroller
- Günstige Komponenten
- Räumlich verteilte Komponenten
- Fernwartung, -diagnose und -test

Vernetzung von Administration und Fertigung



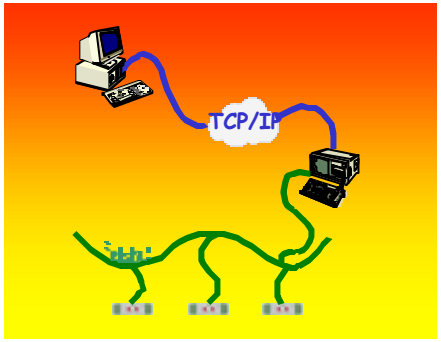
Probleme



Einsatzumfeld

Anwendungen

Verbindungs-
architektur



Anzahl
der
Geräte

Leistungsfähigkeit
der Komponenten

Sicherheits-
anforderungen

Vergangenheit und Zukunft

- Proprietäre Software
- Proprietäre Schnittstellen der einzelnen Komponenten
- Komplettlösungen
- Keine Skalierbarkeit
- Keine Wiederverwendungs-möglichkeiten

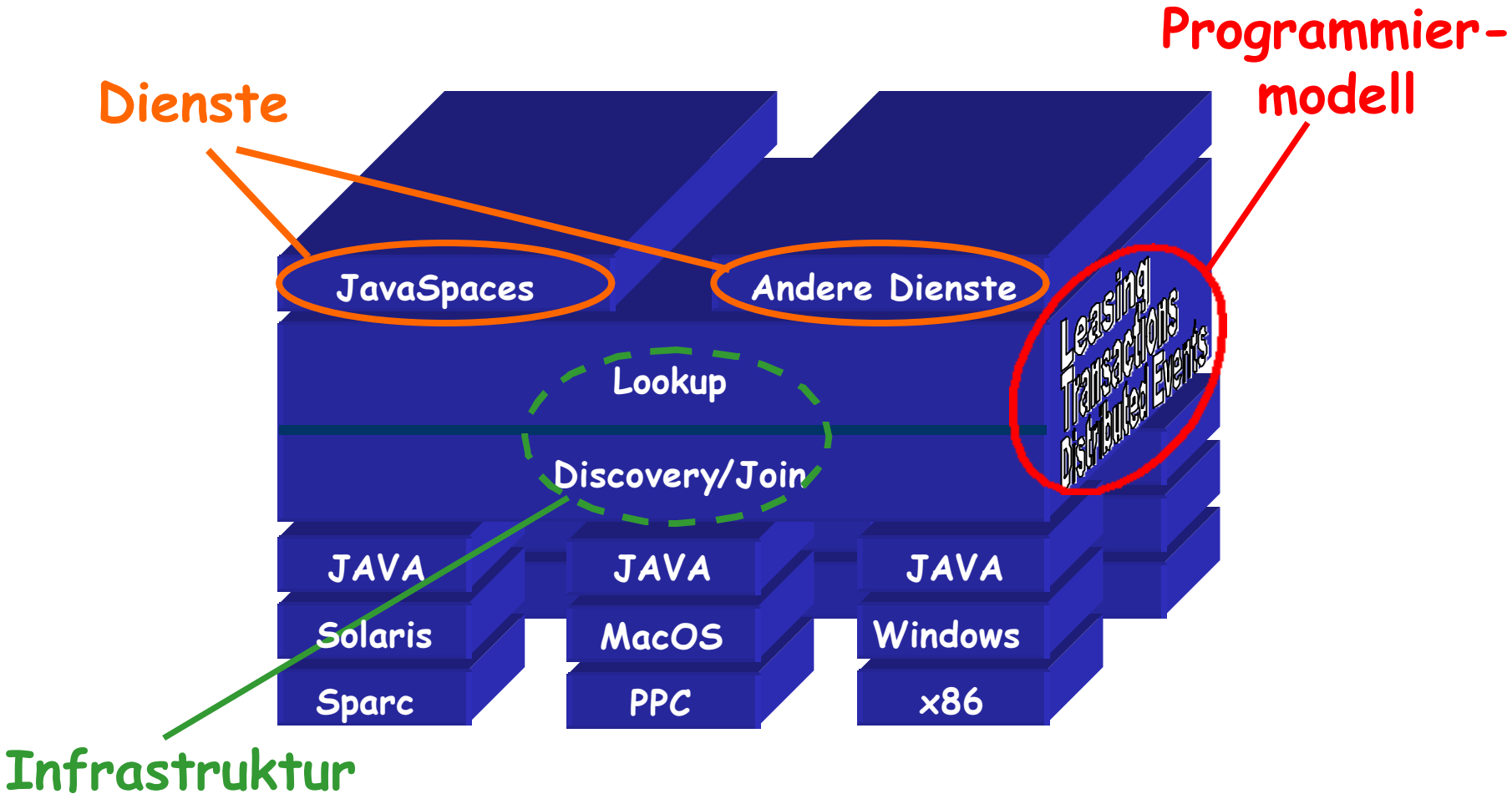
- Durchgängige Software-architektur
- Plug-and-Play Technologie auf allen Ebenen
- Baukastenlösungen
- Skalierbarkeit und Flexibilität



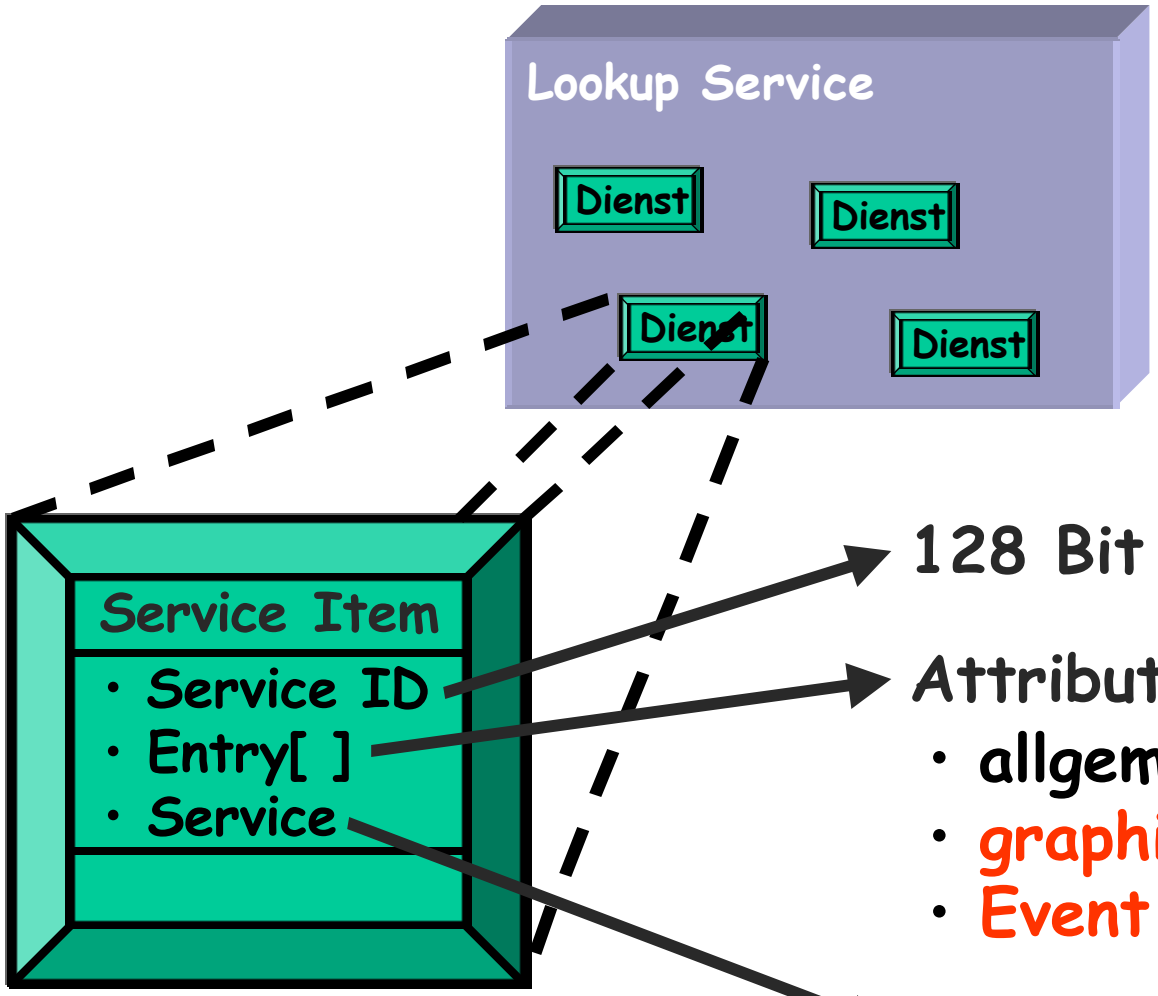
Vorteile einer Jini-Lösung

- Programmiersprache Java mit allen Vorteilen
- Jini stellt Infrastruktur für Plug-and-Play Technologie zur Verfügung
- Jini bietet einheitliche Sichtweise auf unterschiedlichste Komponenten
- Jini-Technologie ist für verteilte Objekte ausgelegt

Jini-Architektur



Repräsentation eines Dienstes



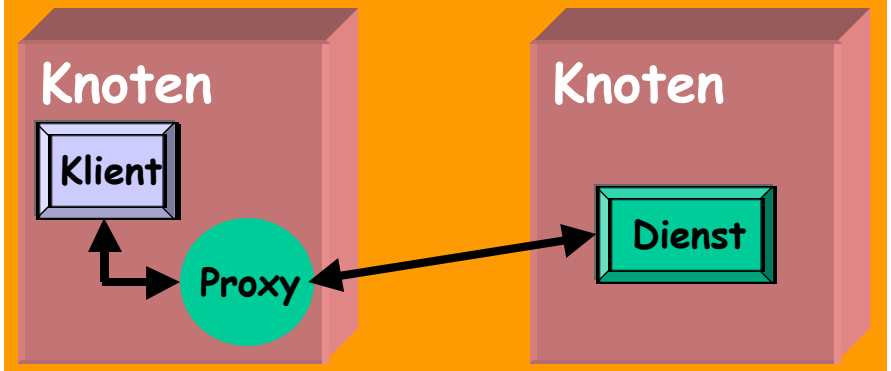
128 Bit Identifier

Attribute des Dienstes
- allgemeine Informationen
- graphische Oberfläche
- Event Handler

Proxy des Dienstes

Auswahl der Systemarchitektur

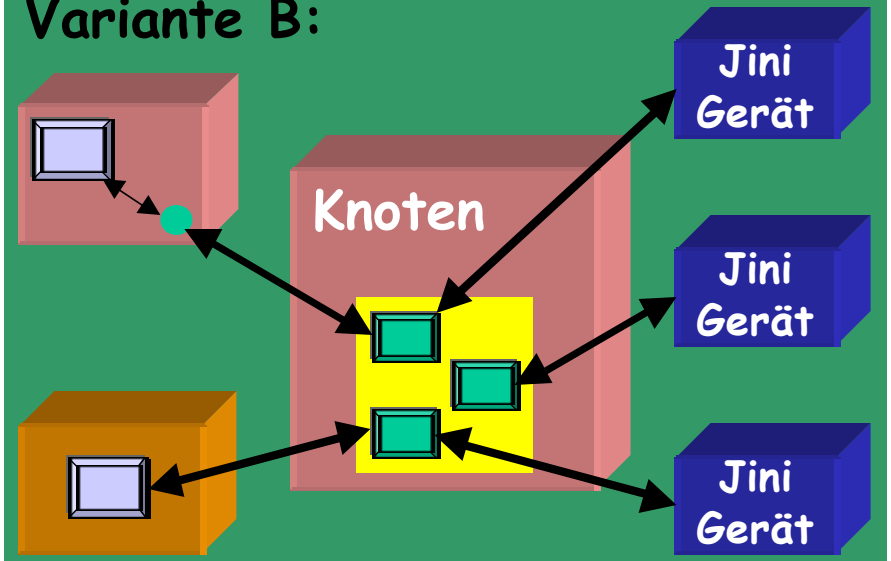
Variante A:



Hohe Systemanforderungen an Klient/Dienst:

- leistungsfähige CPU
- hohe Speicherkapazität
- permanentes Speichermedium

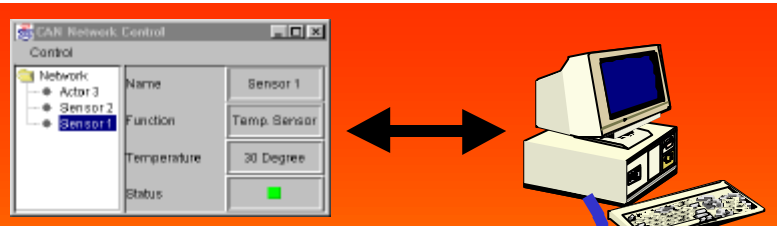
Variante B:



Vorteile:

- Java 2 nur auf einem Knoten
- günstige Geräte im Hinblick auf Speicherbedarf und CPU-Performance

Architektur der Anwendung (HW)



TCP/IP

CAN

Führungsebene

Windows95/NT
Workstation

Steuerungsebene

Windows95
PentiumII

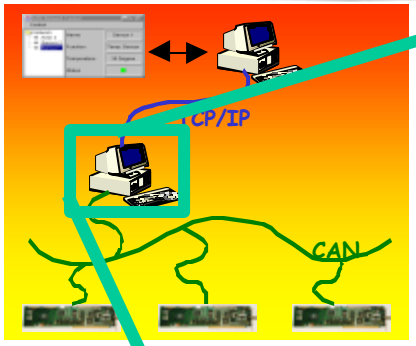
Sensor-/Aktor-ebene

VxWorks
EmbeddedPPC 403

Implementierte Module

- Gerätemanger
  - Anmeldung der Dienste beim Lookup Service
  - Schnittstelle zum CAN-Bus (JNI)
- Event Generator
  - als Dienst implementiert
  - Auslieferung verteilter Ereignisse
- Sensoren/Aktoren
  - als Dienste implementiert
  - Graphische Benutzeroberfläche und Event Handler
- Klient
  - Graphische Benutzeroberfläche
  - Verbindung zum Lookup Service

Architektur der Anwendung (SW)



DeviceManager

Klassen-bibliothek

Sensor

Aktor

Sensor

Sensor
GUI
EH

Aktor
GUI
EH

Sensor
GUI
EH

Lookup Service (Reggie)

Aufgaben des DeviceManagers

1. Anmeldung des EventGenerators

```
EventGeneratorImpl eventGenerator = new EventGeneratorImpl();
JoinManager joinManager = new JoinManager(eventGenerator,
                                          null,
                                         (ServiceIDListener)new DeviceManager(),
                                          new LeaseRenewalManager());
```

2. Anmeldung der Sensoren/Aktoren

```
Device_Impl device = new Device_Impl(name, eventID);
Entry[] attributes = new Entry[2];
attributes[0]      = new SensorInfo();
attributes[1]      = new EventInfo();
JoinManager joinManager3 = new JoinManager(device,
                                           attributes,
                                           (ServiceIDListener)new DeviceManager(),
                                           new LeaseRenewalManager());
```

3. Schnittstelle zum CAN-Bussystem

- Zugriff auf ISA-CAN-Karte über JNI
- Synchronisation des CAN-Zugriffs

Anfragen des Klienten

1. Auffinden des Lookup Service

```
LookupLocator locator = new LookupLocator(jini://nautilus);
```

2. Zeiger auf den Service Registrar

```
ServiceRegistrar sr = locator.getRegistrar();
```

3. Anlegen eines Suchmusters

```
Class[] classes = new Class[] { Device.class };
ServiceTemplate st = new ServiceTemplate(null,      // ID
                                         classes,   // Typen
                                         null);     // Attribute
```

4. Anfordern der „Dienste"

```
ServiceMatches sm = sr.lookup(st, 5);
ServiceItem[] si = new ServiceItem[sm.totalMatches];
si[i] = sm.items[i];
```

5. Instanziierung der Proxies

```
Device dev = (Device) si[i].service;
```

Auflösung der Attribute von Diensten

Attribut

```
class GUIEntry
      implements Entry {
  public
  public name
  public

  public GUIEntry(...)
  {...}
 }
```



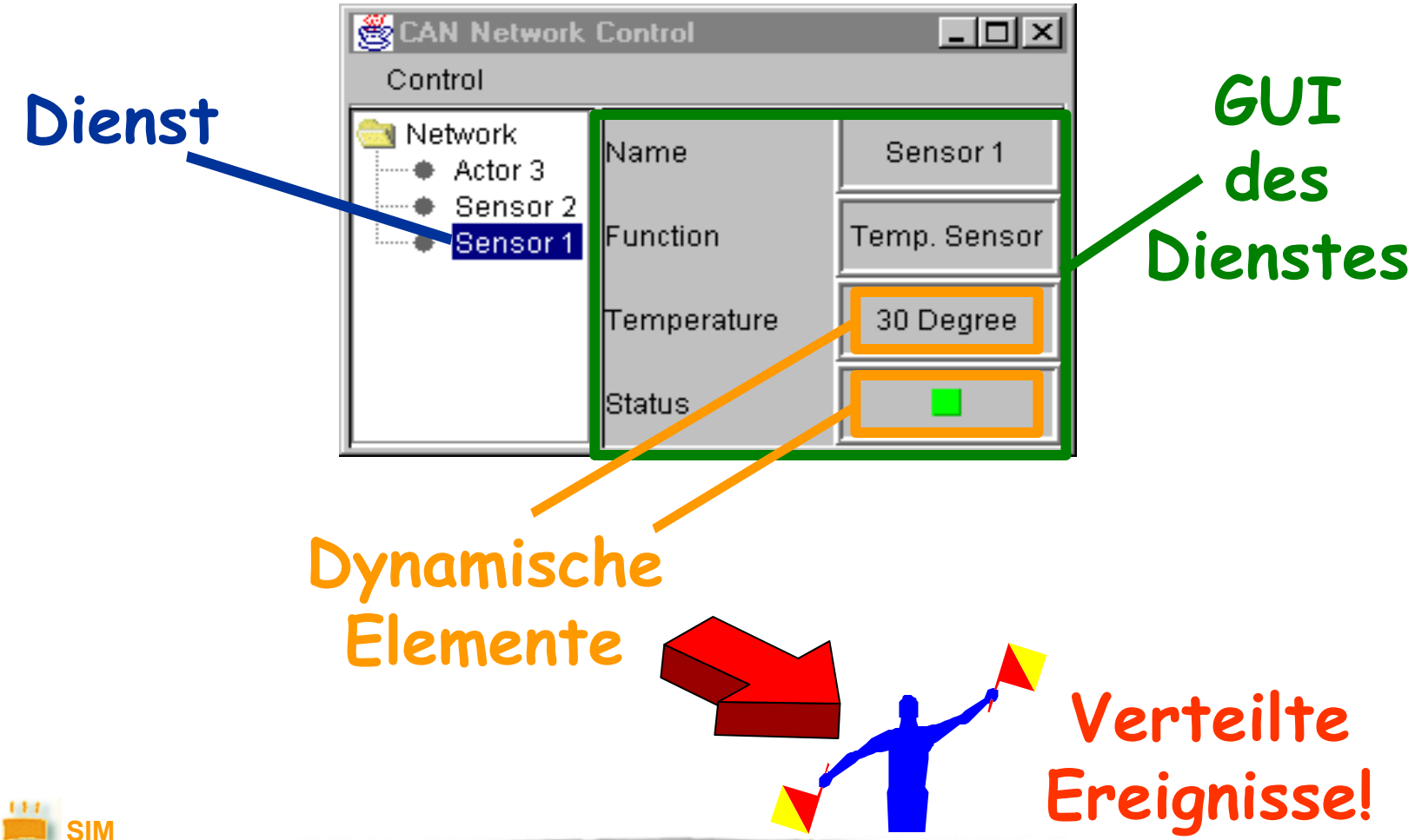

```
e = sr.getEntryClass()
f = e.getField(name)
c = f.getType()
i = c.newInstance()
```

```
e = sr.getEntryClass()
f = e.getField(name)
c = f.getType()
i = c.newInstance()
```

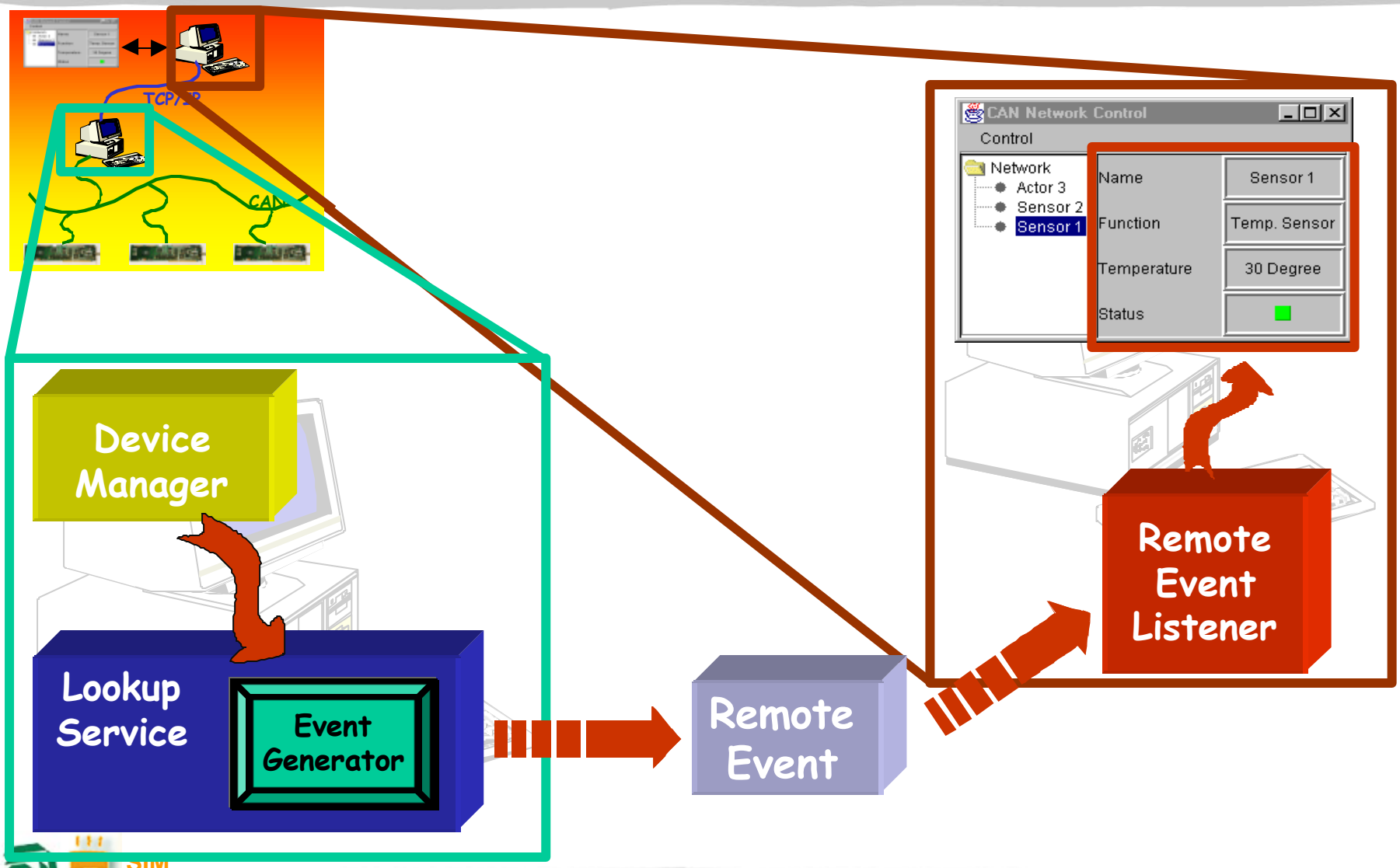
```
e = sr.getEntryClass()
f = e.getField(name)
c = f.getType()
i = c.newInstance()
```

Benutzeroberfläche des Klienten

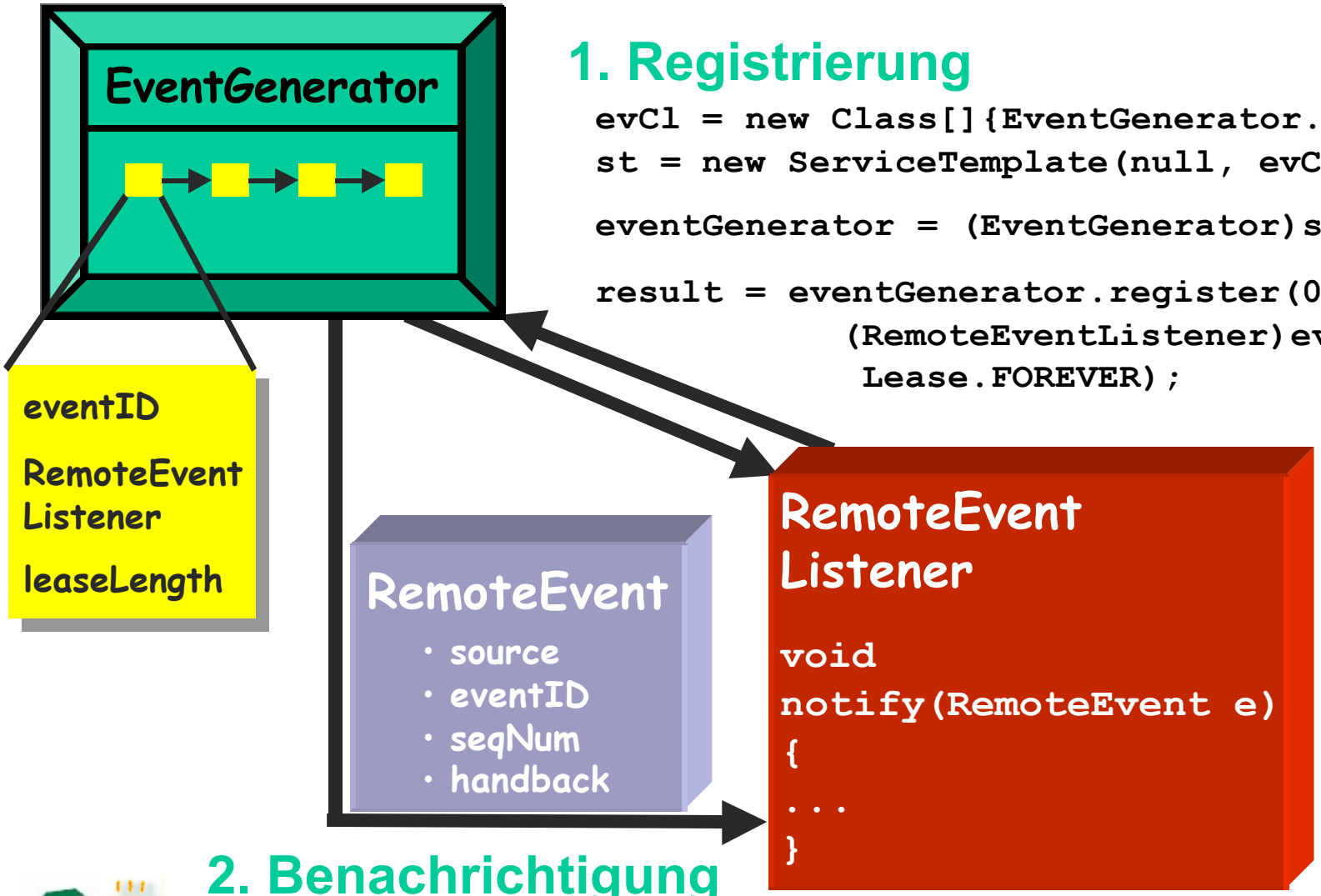
➡ **Realisiert mit dem JFC Swing-Paket**



Verteilte Ereignisbehandlung



Implementierung verteilter Ereignisse



1. Registrierung

```
evCl = new Class[]{EventGenerator.class};
st = new ServiceTemplate(null, evCl,null);

eventGenerator = (EventGenerator)sr.lookup(st);

result = eventGenerator.register(0,
            (RemoteEventListener)eventListener,
             Lease.FOREVER);
```

2. Benachrichtigung

Ergebnisse

Aufgaben von Jini:

- Reggie (Lookup Service)
- HTTP-Server
- Kommunikation
- Rahmenklassen für die Dienste



Aufgaben des Entwicklers:

- DeviceManager
- Ereignisbehandlung
- Benutzeroberflächen
- Anbindung an das Feldbussystem



- Konzentration auf die Anwendung
- Komponentenbasierte Anwendungsentwicklung
- Plug-and-Play Technologie
- Umfangreiche Infrastruktur vorhanden

Arbeitsaufwand

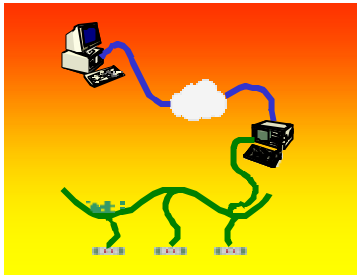
	Steuerungsebene			Führungs-ebene
	Device Manager	Event Generator	Dienste	Klient
Anzahl Klassen insgesamt	1+1	1	9	8
Klassen mit Jini-Bezug	1	1	4	1
LOC	131+116	79	388	622

Probleme im Zusammenhang mit Jini

- Java 2 Plattform auf der Steuerungsebene
  - Hardware:
    PentiumII mit 128MB RAM
  - Software:
    Windows95
  - Lösungmöglichkeit:
    J2ME soll Jini unterstützen
- Echtzeitunterstützung in Jini?
  - Reaktion auf verteilte Ereignisse
  - Auslieferung verteilter Ereignisse

Zusammenfassung

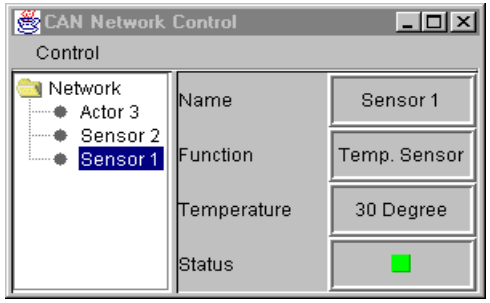
- Anforderungen an eingebettete Systeme steigen



- Jini bietet Infrastruktur für verteilte Systeme an



- Anwendung der Jini-Technologie zur Überwachung eines CAN-Netzwerkes



Ausblick

Ethernettechnologie auf der Sensor-/Aktorebene

- physikalisch
- logisch

Java auf der Sensor-/Aktorebene

- KVM und J2ME
- Verteilte JVM
- Java Compiler
- Jini on a Chip