Prototypimplementierung der Real-Time Data Access (RTDA) Spezifikation 2.0 für den Java-Hardwarezugriff am Beispiel einer Controller Area Network (CAN) Schnittstellenkarte

Studienarbeit

01.10.2005-31.12.2005

Cand. Inform. Alexander Roth
01728892838@vodafone.de

Inhalt

- Einleitung
- Grundlagen
- RTDA 2.0
- Implementierung
- Ausblick

Einleitung

- Treiberprogrammierung (in ES)
  - Hardwareplattformabhängig
  - Betriebssystemabhängig
- Treiberprogrammierung in modernen BS
  - Hardwareplattformunabhängig
  - Betriebssystemabhängig

→ Programmierung mit plattformunabhängigen Sprachen
  - Java

Grundlagen

- Mit Java kein Hardwarezugriff möglich

→ Kapselung nativer Bibliotheken

  - Java Native Interface (JNI)
    - Aufwändig
    - Plattformabhängig
  - JNI-Wrapper
    - Weniger aufwändig
    - Noch immer plattformabhängig

Grundlagen

- Real-Time Data Access (RTDA) Version 1.0
  - Rahmenwerk für die Treiberprogrammierung
    - Auf Basis von JNI
      - plattformabhängige Treiber weiterhin nötig
    - Keine Kompatibilität mit existierenden Java-Bibliotheken
    - Wenig objektorientiert

→ Anpassung an RTSJ (Real-Time Specification for Java)

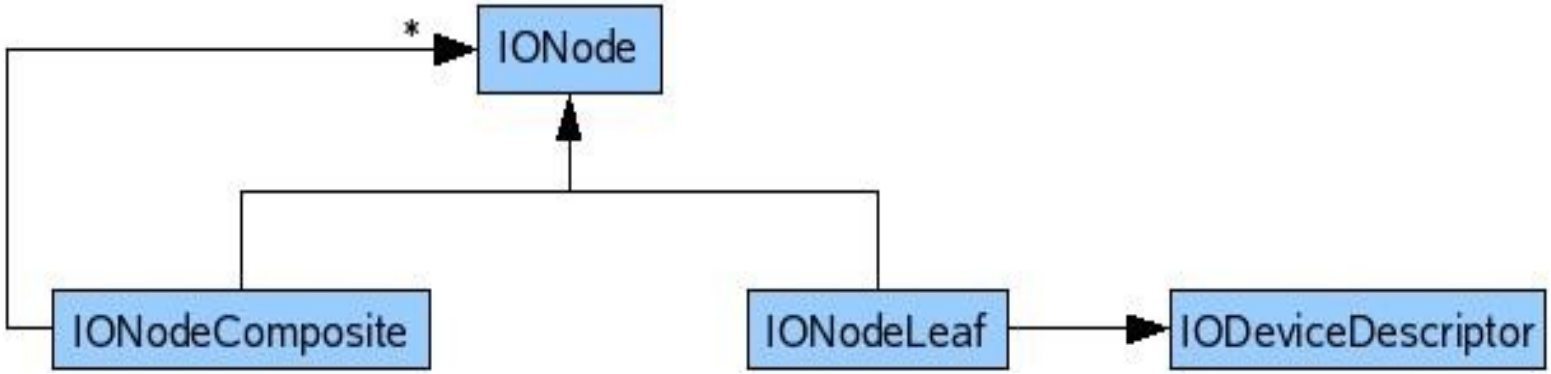
→ Nutzung objektorientierter Entwurfsmuster

RTDA 2.0

- Übersicht verwendeter Konzepte
  - Entwurfsmuster Kompositum
  - Implementierungsunabh. Gerätebeschreibung
  - E/A-Kanäle für den Speicherzugriff
  - Ereignisbehandlung

RTDA 2.0: Kompositum

- IONodeComposite
  - beschreibt den Zugriffspfad zu einer Gerätebeschreibung
- IONodeLeaf
  - referenziert eine Gerätebeschreibung



RTDA 2.0: Gerätebeschreibung

- Gerätebeschreibung (IODeviceDescriptor)
  - Information über aktuelle Hardwarekonfiguration
    - Für Konstruktion von E/A Kanälen
    - Für Unterbrechungsbehandlung
- Erweiterung im Rahmenwerk
  - Hersteller- und Geräteinformation
  - Bus-Informationen

→ Eindeutige Gerätezuordnung

RTDA 2.0: E/A Kanäle

- Konstruiert mit Eintrag aus IODeviceDescriptor
- Schreiben/Lesen von Daten
  - IOByte[Array]
  - IOShort[Array]
  - IOInteger[Array]
  - IOLong[Array]

→ Implementiert mit RawMemoryAccess von RTSJ

RTDA 2.0: Ereignisbehandlung

- Ereignistypen
  - unterbrechende (InterruptEvent)

→ Implementiert mit AsyncEvent von RTSJ

  - periodische (PeriodicEvent)
  - unregelmässige (SporadicEvent)
  - einmal auftretende (OneShotEvent)

- Ereignis-Behandler
  - IOEventHandler

→ Implementiert mit AsyncEventHandler von RTSJ

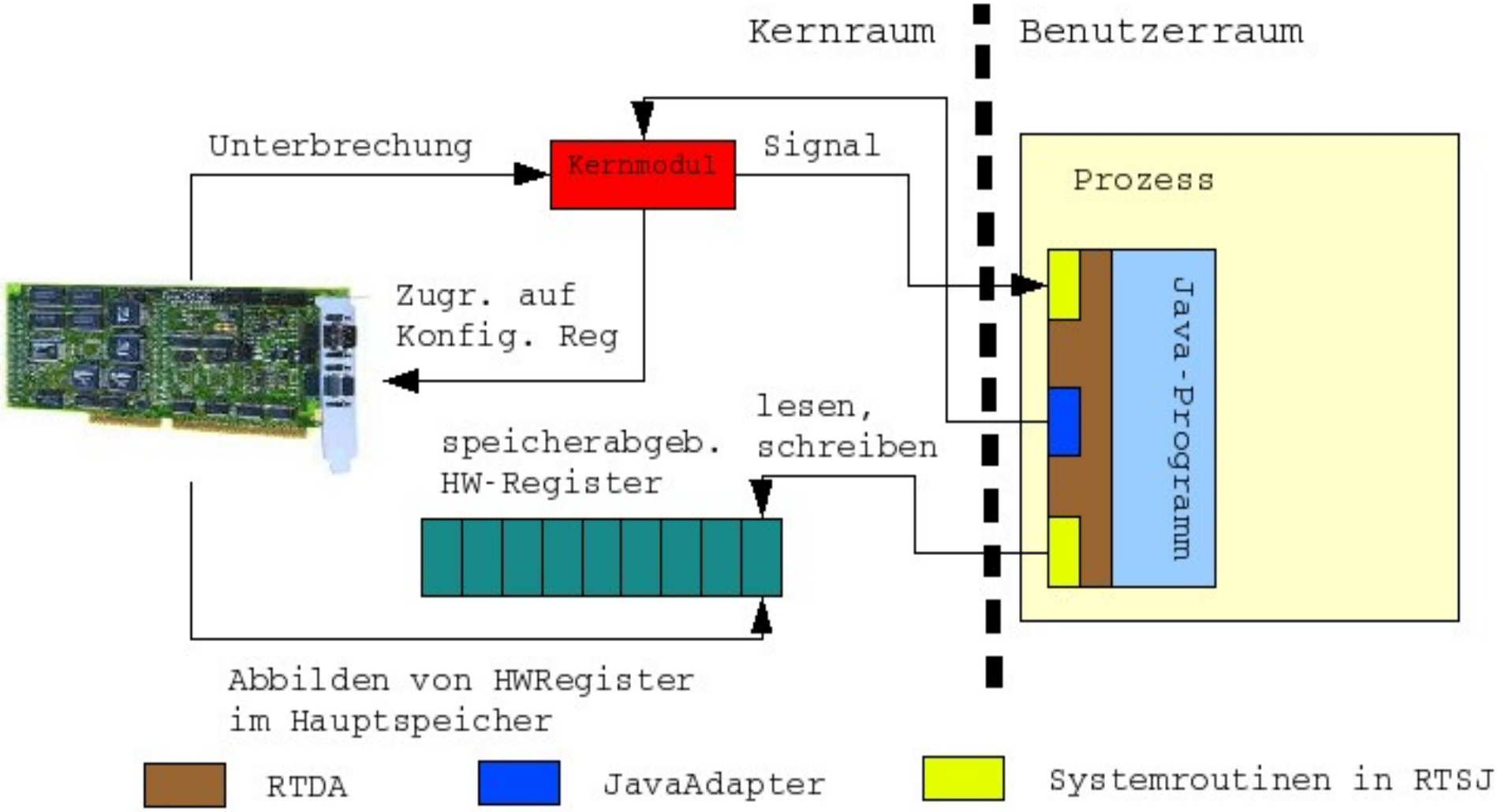
Kernmodul

- Abbildung Hardwareunterbrechung auf Signal
- Liefert Informationen über Geräte im System
- Zugriff auf PCI Register
- Zugriff auf Gerätekontrollregister

Java Adapter

- Java Schnittstelle zum Linux-Kern
  - Leitet Befehle des Java-Treibers an das Kernmodul
  - Zugriff auf PCI Register
  - Zugriff auf Gerätekontrollregister

Architektur des Rahmenwerks



Beispiel Java-Treiber

- Erzeuge dynamisch das Kompositum

```
IONodeComposite root = new IONodeComposite(„rtda“, „/proc/hwdescriptor“);
IONodeLeaf ionl = new IONodeLeaf(root, „0x10b5“, „0x9050“, „0x10b5“,
„0x1067“);
```

- Finde ein IODeviceDescriptor

```
IODeviceDescriptor ddescr = ionl.getIODeviceDescriptor();
```

- Erzeugen des E/A Kanals

```
InterruptEventContext iec = new InterruptEventContext(ddescr, new
RealtimeThread());
IOInteger ioint = iec.createIOInteger((short)2);
```

- Anbindung einer Unterbrechung

```
InterruptEvent ie = iec.createInterrupt((KernelInterface) jadapt);
```

- Zugriff auf Hardware

```
int intval = ioint.read(0x00);
```

Fragen