Plug & Care Connector: OSGi-basierte Applikation für Smartphones und Desktop-Systeme

Doreen Seider

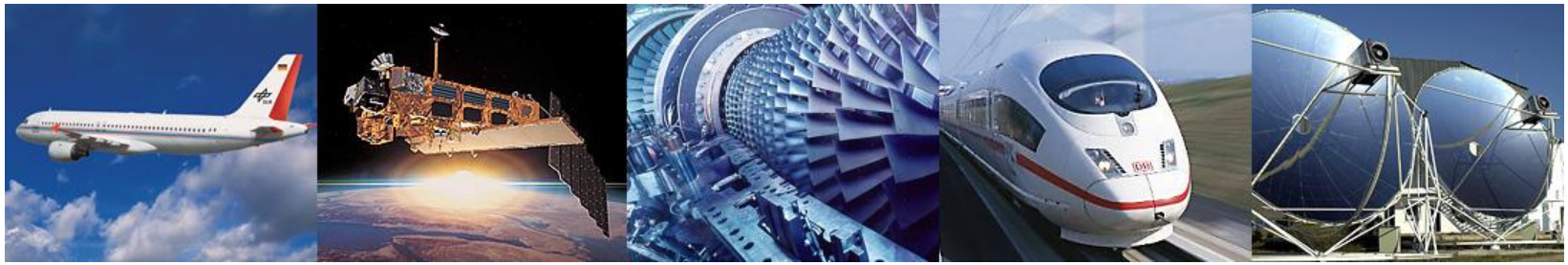
Deutsches Zentrum für Luft- und Raumfahrt (DLR)

BerlinExpertDays
Berlin, 07.04.2011

Roter Faden

- Hintergrund: EU-Projekt „EmotionAAL“
- Idee des Plug&Care Connectors
- Realisierung mit OSGi
- Best Practices, Probleme, Ausblick

Das DLR
Deutsches Zentrum für Luft- und Raumfahrt



- Forschungseinrichtung
- Raumfahrt-Agentur
- Projektträger

Hintergrund: EU-Projekt „EmotionAAL“

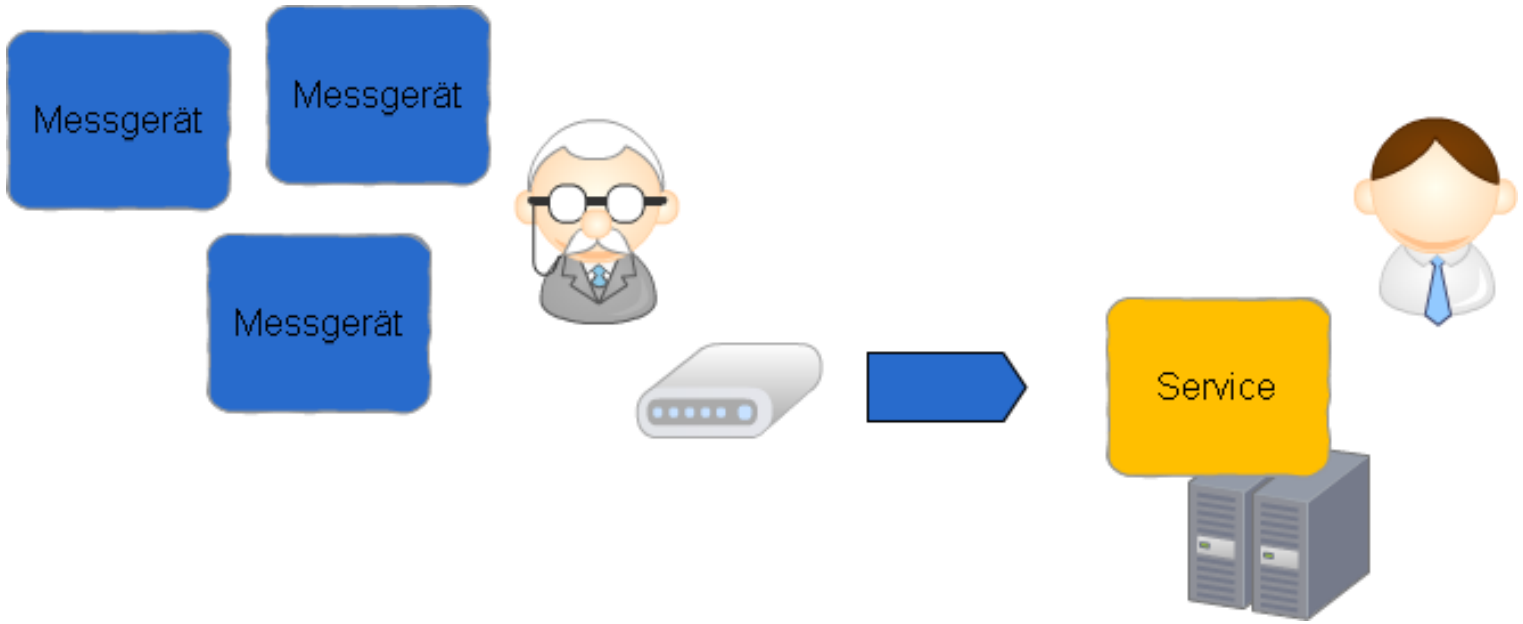
„Support of people with chronical diseases in rural regions“

- Ambient Assistent Living (AAL)
- 10 Projektpartner aus Deutschland, Österreich und Finnland
- Laufzeit: Juli 2009 – Juli 2012
- DLR: Simulations- und Softwaretechnologie, Institut für Raumfahrtmedizin

Hintergrund: EU-Projekt „EmotionAAL“

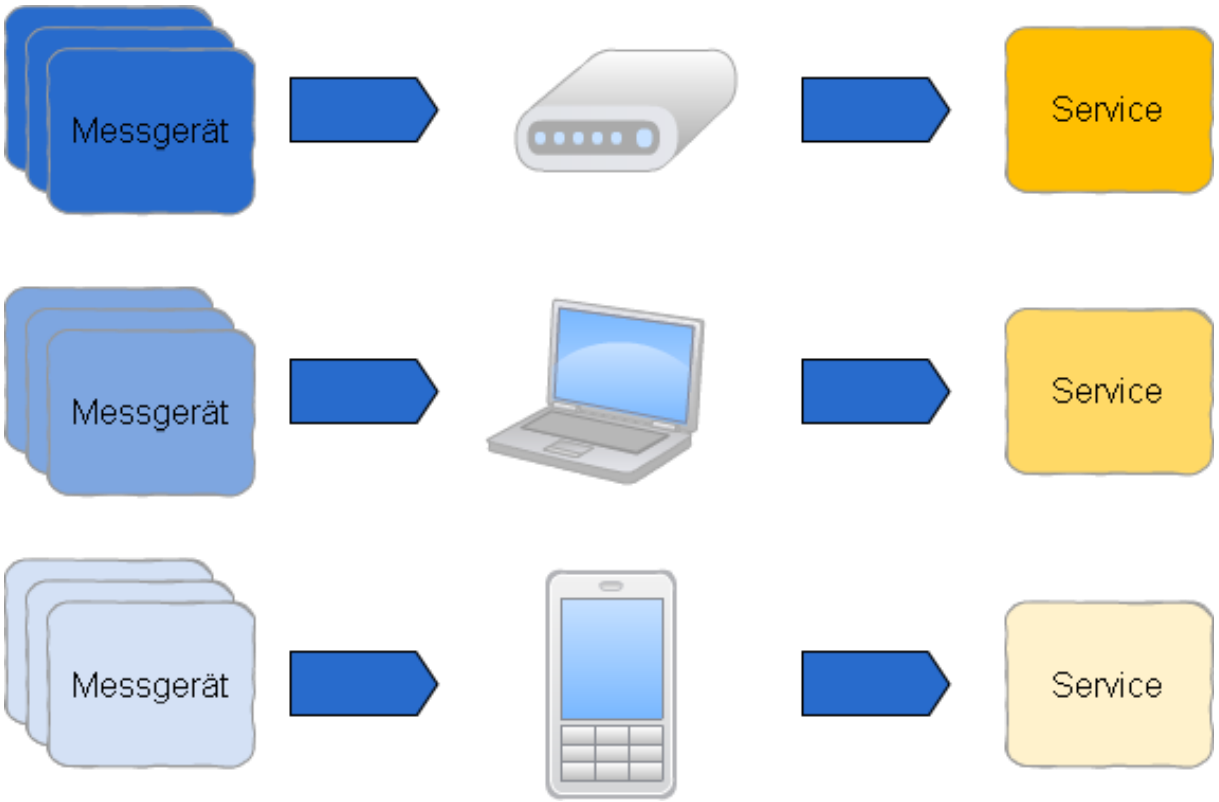
- Eines der Ziele: Flexibles Telemonitoring-Setup

Unflexible Telemonitoring-Setups

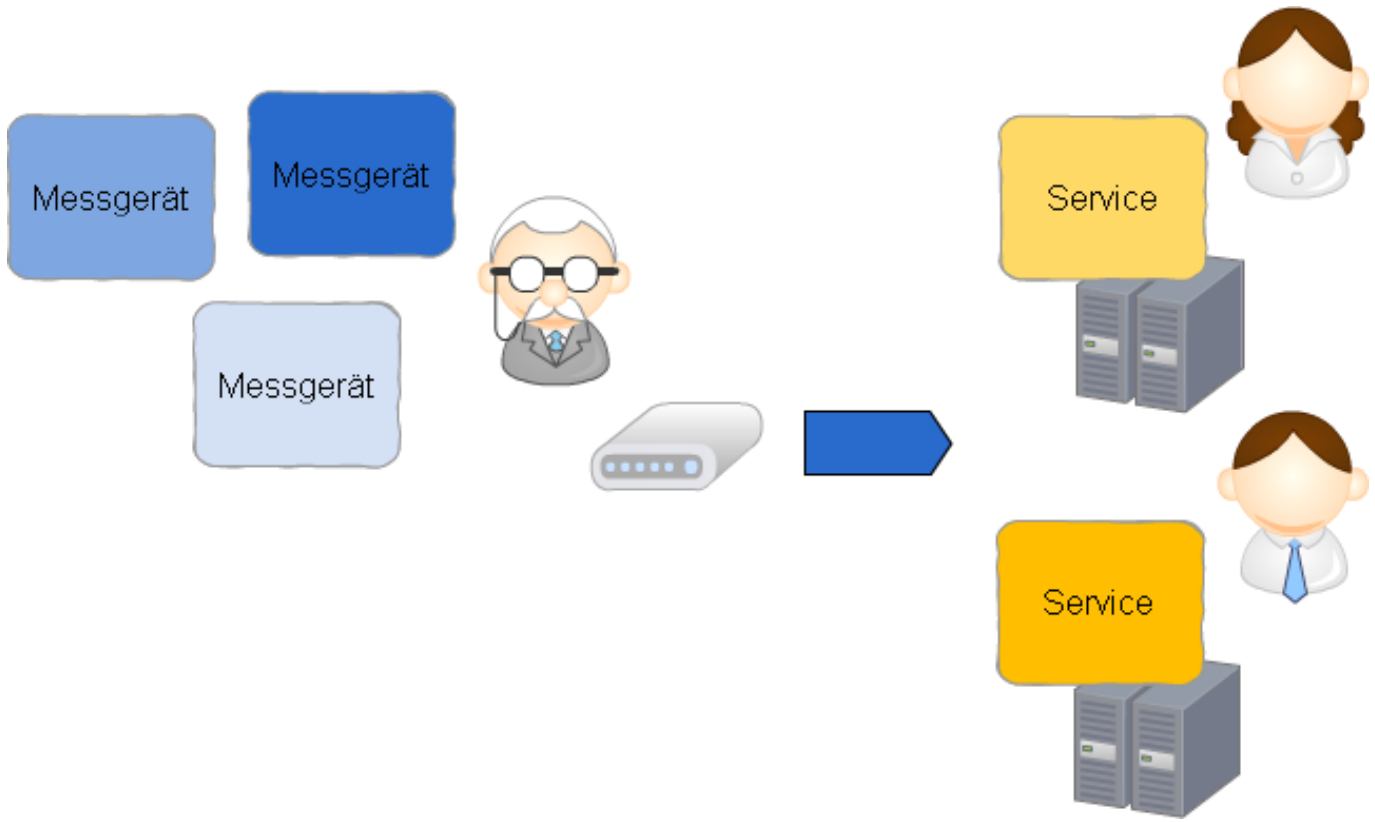


Unflexible Telemonitoring-Setups

Connector-Technologien

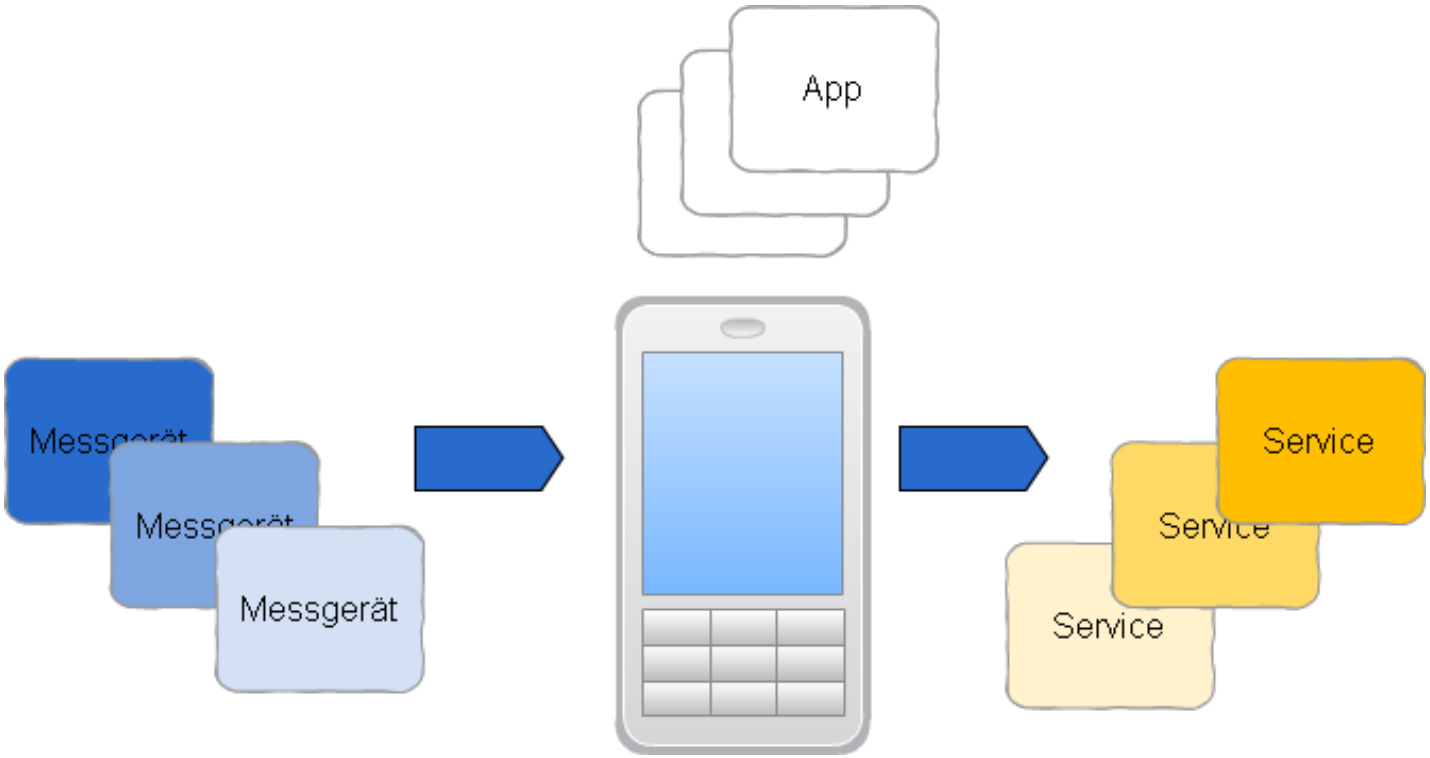


Flexibles Telemonitoring-Setup



Flexibles Telemonitoring-Setup

Connector-Technologie: Plug&Care Connector



Plug&Care Connector

Anforderungen

1. Unterstützung beliebiger Messgeräte und Expertenzentren (auch im Nachhinein) ohne Plug&Care Connector anzupassen

2. Sowohl auf Smartphones (wie Android oder Windows Mobile) als auch auf Desktop-Betriebssystemen lauffähig

Plug&Care Connector

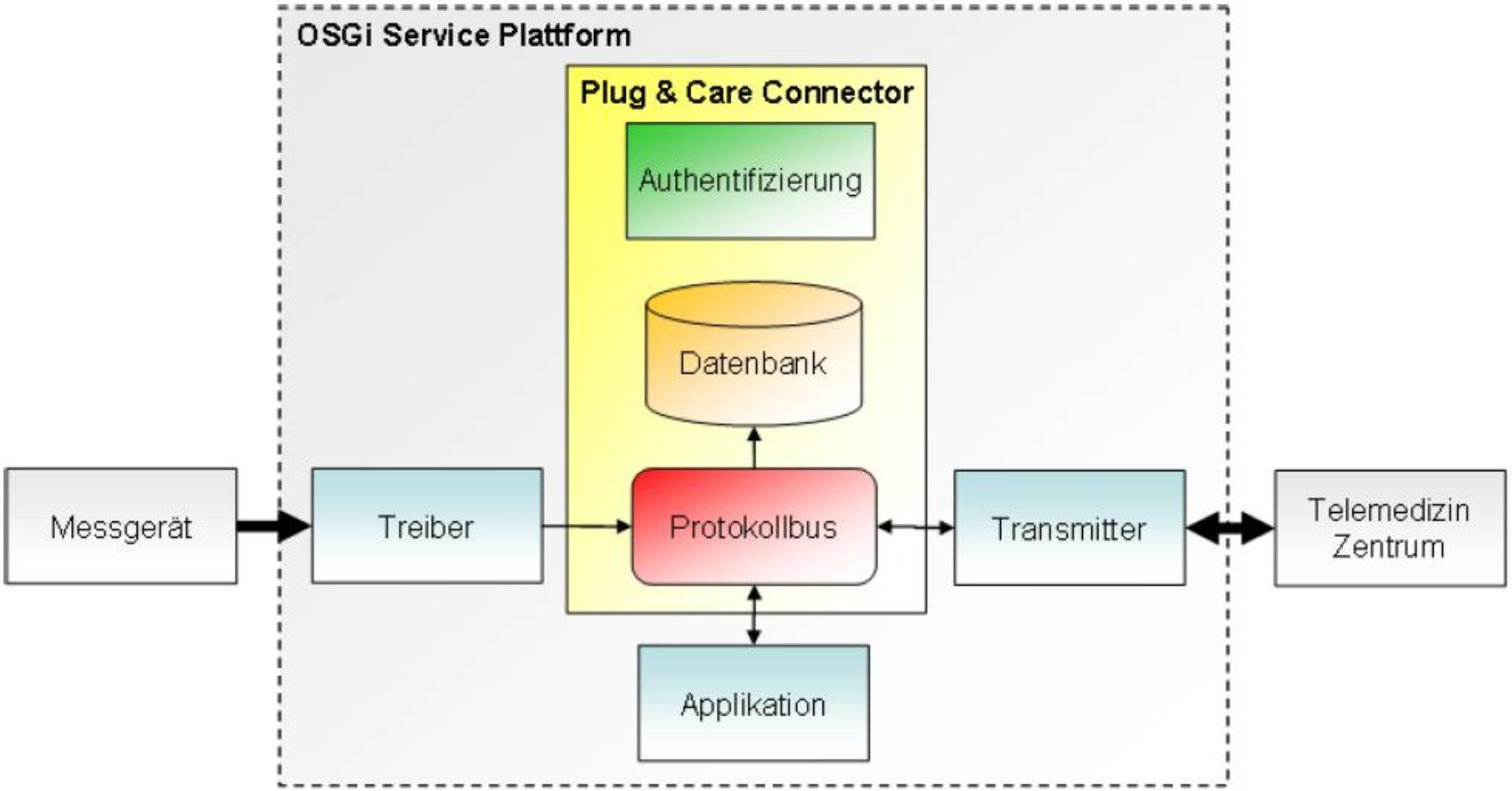
Erfüllen der Anforderungen

1. Plugin-Konzept auf Basis von
   - Modularität von OSGi
   - Dependency Injection durch OSGi Declarative Services

2. Plattformunabhängigkeit durch Realisierung als OSGi-Applikation
   - Equinox auf Desktop
   - mBS Mobile auf Smartphone

Plug&Care Connector

Architektur



OSGi-Stack: Equinox

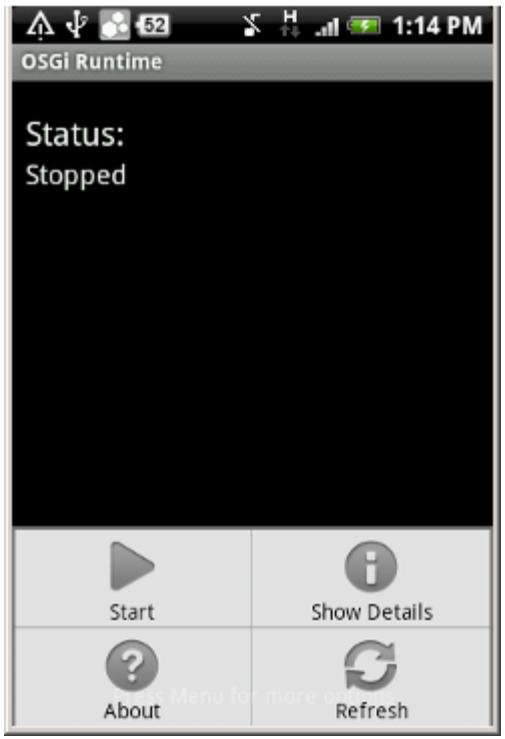
- Eclipse Software Foundation
- R4 Core und Service Compendium und optionale OSGi Services
- OSGi-Implementierung für Eclipse

OSGi-Stack: mBS Mobile

- Firma ProSyst (http://prosyst.com)
- Android, Windows Mobile, Nokia S60
- Leicht unterschiedlicher Funktionsumfang pro Plattform
- Allen gleich: alle OSGi-Services und Framework-Features spezifiziert in JSR 232 und einige nicht spezifizierte OSGi-Services (HTTP mit JSP 2.0, User Admin, …)
- Übersicht der Stacks: http://dz.prosyst.com/mbsmobile/

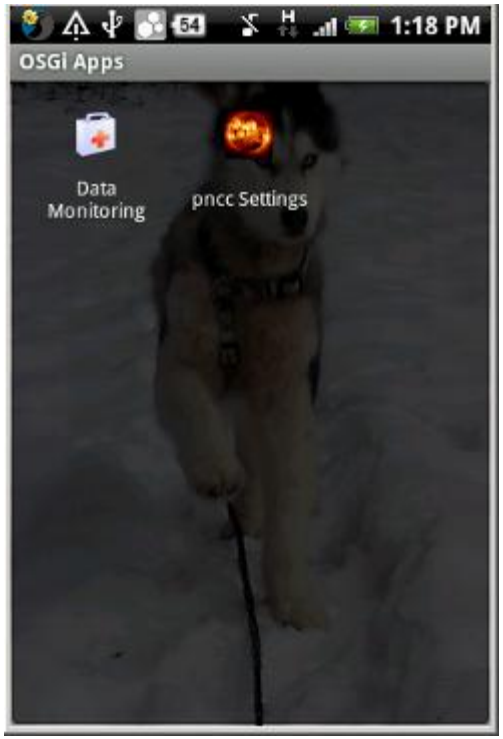
OSGi-Stack: mBS Mobile

OSGi Management



OSGi-Stack: mBS Mobile

OSGi Applications



OSGi-Stack: mBS Mobile

Web-Widgets

- Allgemein
  - Clientseitige Applikationen
  - Eigenständig oder Web-Browser-basiert
  - GUI in HTML
  - GUI- und Geschäftslogik in JavaScript

- Konzept in mBS Mobile
  - GUI für OSGi-Applikationen
  - Web-Browser-basiert
  - GUI in HTML
  - GUI-Logik in JavaScript
  - Geschäftslogik in Java (OSGi-Services)

OSGi-Stack: mBS Mobile

Web-Widgets

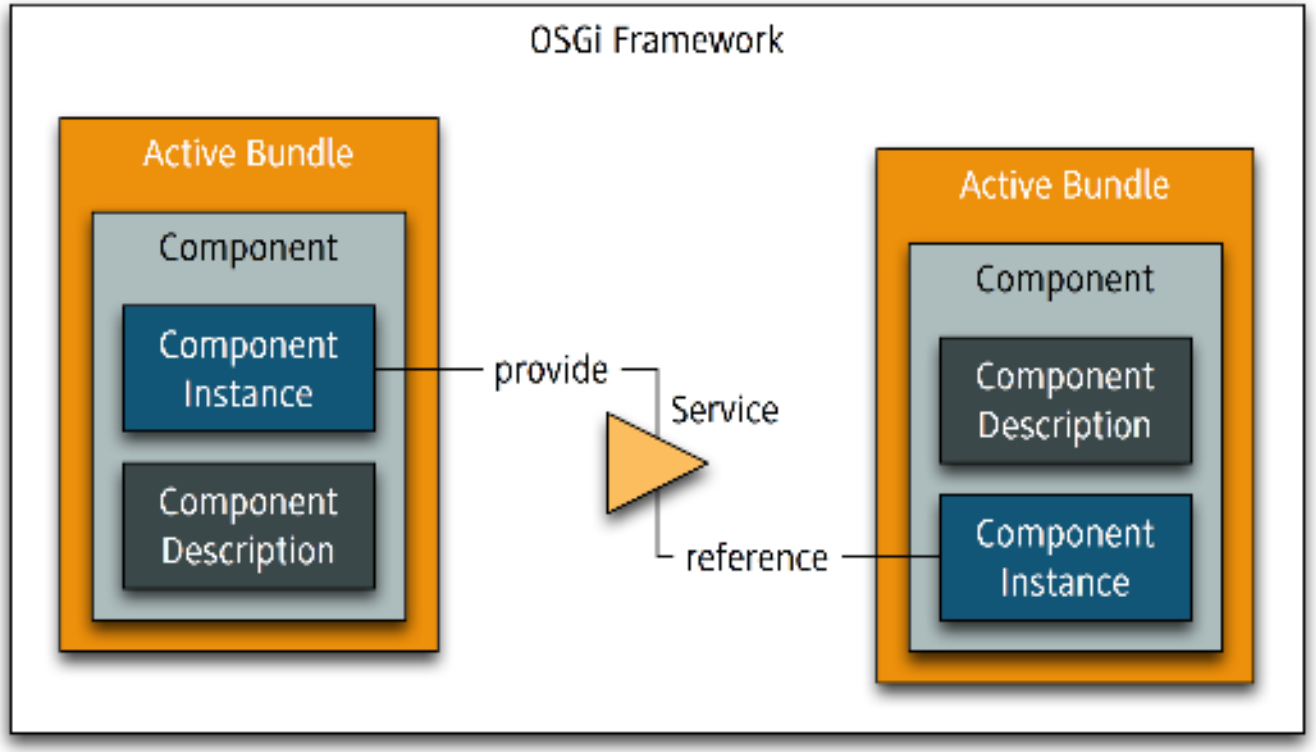
- Bereitstellen von OSGi-Services
  - Angelehnt an Distributed-OSGi-Konzept
  - Deklarieren von OSGi-Services als „exported“

- JavaScript-Bibliothek RSR (mBS Mobile)
  - Finden und Binden von exportierten OSGi-Services
  - Verwenden der OSGi-Services als JavaScript-Objekte

Exkurs: Declarative Services (DS)

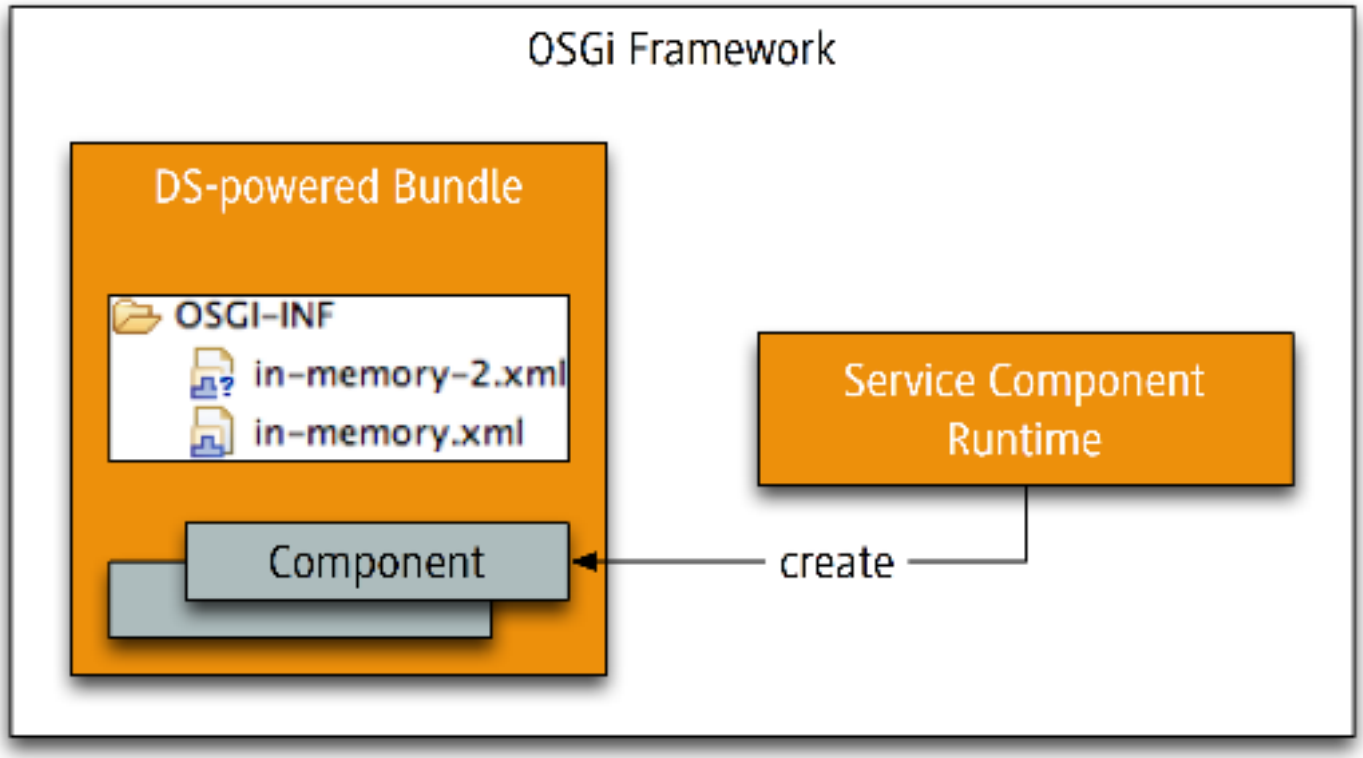
- Beschreiben der Service-Abhängigkeiten zwischen Bundles
- Handling der Abhängigkeiten durch DS Runtime
- Vorteile
  - Geringere Komplexität (Dynamik)
  - Geringere Start-up-Zeiten

Exkurs: Declarative Services (DS)



Quelle: Heiko Seeberger

Exkurs: Declarative Services (DS)



Quelle: Heiko Seeberger

Exkurs: Declarative Services (DS)

- DS Component Description (minimal)

```
<component name=„Spitzen Tool">
    <implementation class=„de.dlr.tool.internal.SpitzenImpl"/>
</component>
```

- Beispiele zusätzlicher Eigenschaften:

```
    <service>
        <provide interface="de.dlr.tool.Spitzen"/>
    </service>

    <property name=„de.dlr.tool.unit">meter</property>
```

OSGi-Stack: mBS Mobile

Web-Widgets

- Deklarieren von OSGi-Services als „exported“
- DS Component Description (vom Widget-Service):

```
<component name=„Widget Service>
    <implementation class="de.pncc.application.internal.WidgetServiceImpl"/>
    <service>
        <provide interface="de.pncc.application.InternalWidgetService"/>
    </service>
    <property name="org.osgi.remote.publish">true</property>
</component>
```

OSGi-Stack: mBS Mobile

Web-Widgets

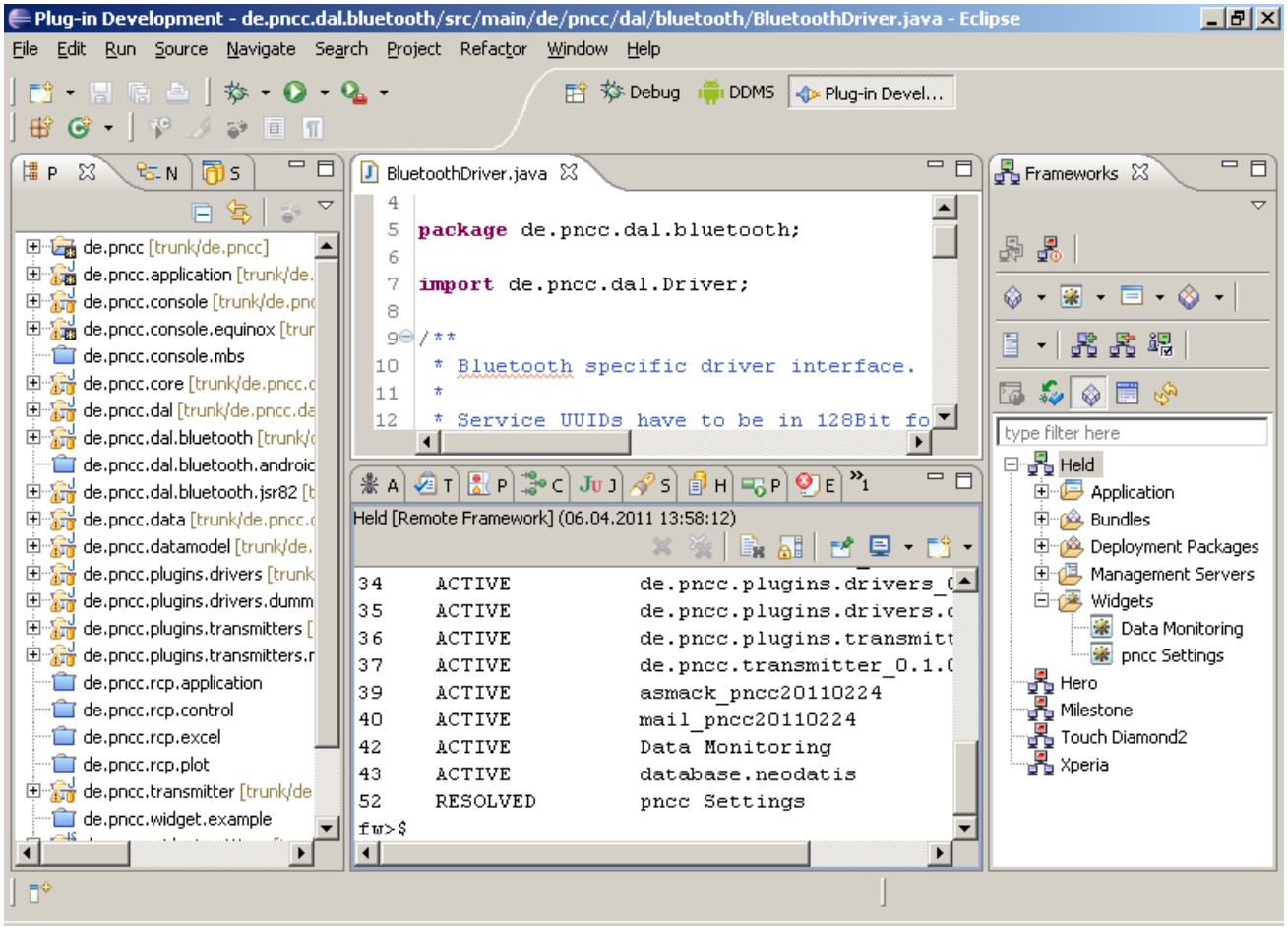
- Verwenden der OSGi-Services als JavaScript-Objekte
- JavaScript-Code in einem Widget:

```
var widgetService = RSR.bind(„de.pncc.application.WidgetService“);

if (widgetService.isLoggedIn()) {
    // do something
}
```

OSGi-Stack: mBS Mobile

SDK



Architektur Plug&Care Connector

Treiber- und Transmitter-Plugins

- Anforderungen
  - Autarkes Deployment
  - Driver/Transmitter als OSGi Services
  - Mehrere Instanzen auf Grund verschiedener Konfigurationen
  - OSGi so transparent wie möglich

- Konzepte
  - OSGi Declarative Services (DS)
  - DS Component Factory

Architektur Plug&Care Connector

Plugin-Konzept am Beispiel von Treiber-Plugins

- Jeder Treiber als DS Component Factory deklariert
- DS Component Description (von Driver B):

```
<component name=„Driver B“ factory="de.pncc.driver">
    <implementation class="de.pncc.plugins.drivers.DriverB"/>
    <service>
        <provide interface="de.pncc.drivers.Driver"/>
    </service>
</component>
```

Architektur Plug&Care Connector

Plugin-Konzept am Beispiel von Treiber-Plugins

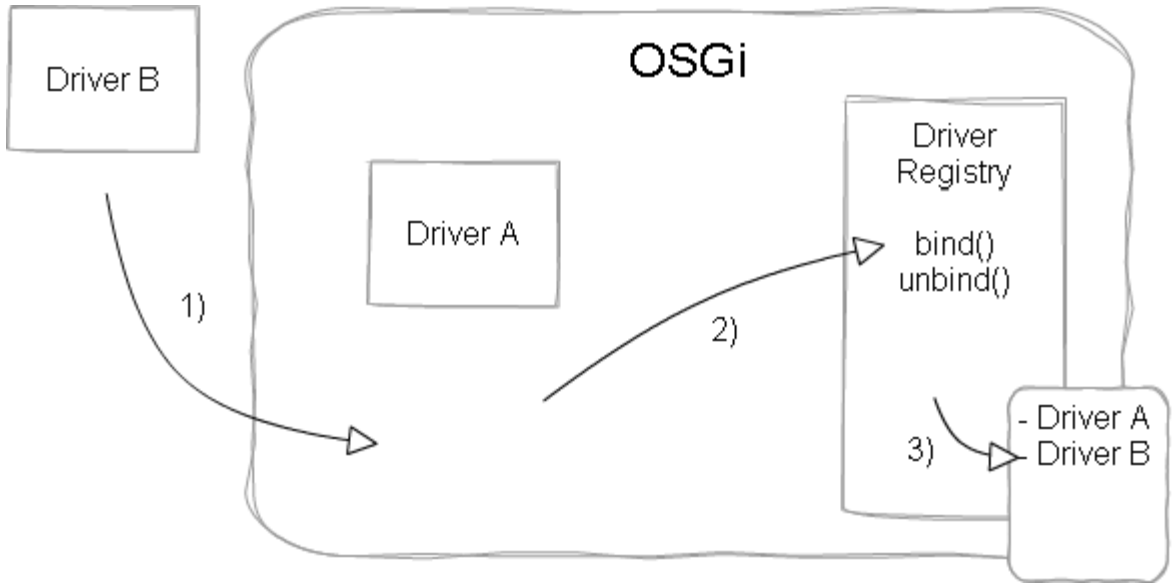
- Verwalten von installierten Treibern: Driver Registry
- DS Component
- Injezieren von Component Factorys vom Typ: „de.pncc.driver“

```
<component name=„Driver Registry„
    <reference
        name="de.pncc.driver„
        interface="org.osgi.service.component.ComponentFactory„
        bind="addDriver„
        unbind="removeDriver„
        target="(component.factory=de.pncc.driver)"/>
</component>
```

Architektur Plug&Care Connector

Plugin-Konzept am Beispiel von Treiber-Plugins

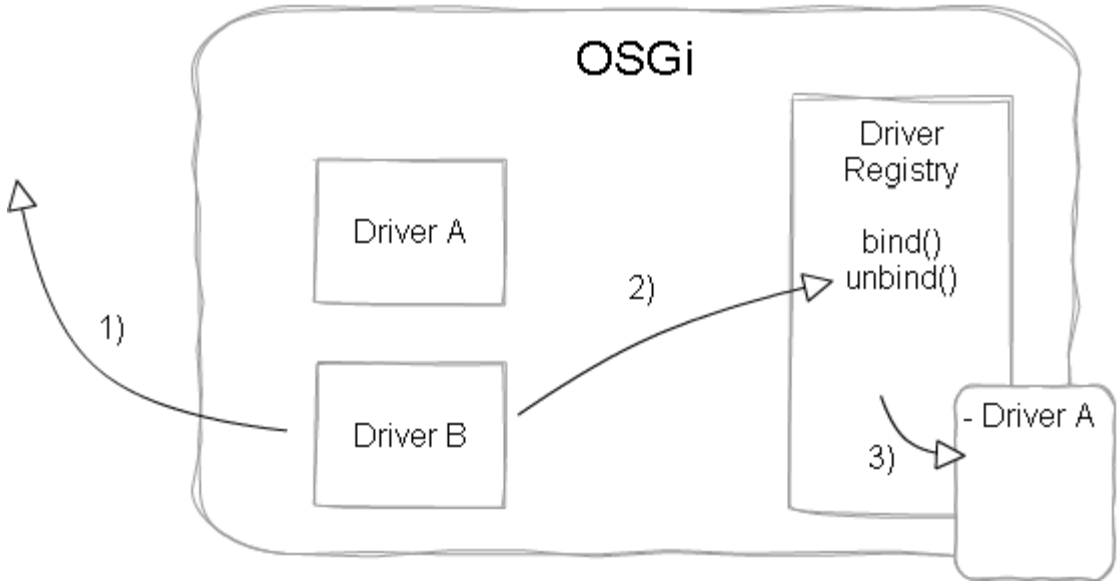
- OSGi-Mechanismen bei Installation (Deployment)



Architektur Plug&Care Connector

Plugin-Konzept am Beispiel von Treiber-Plugins

- OSGi-Mechanismen bei Deinstallation



Architektur Plug&Care Connector

Datenhandling

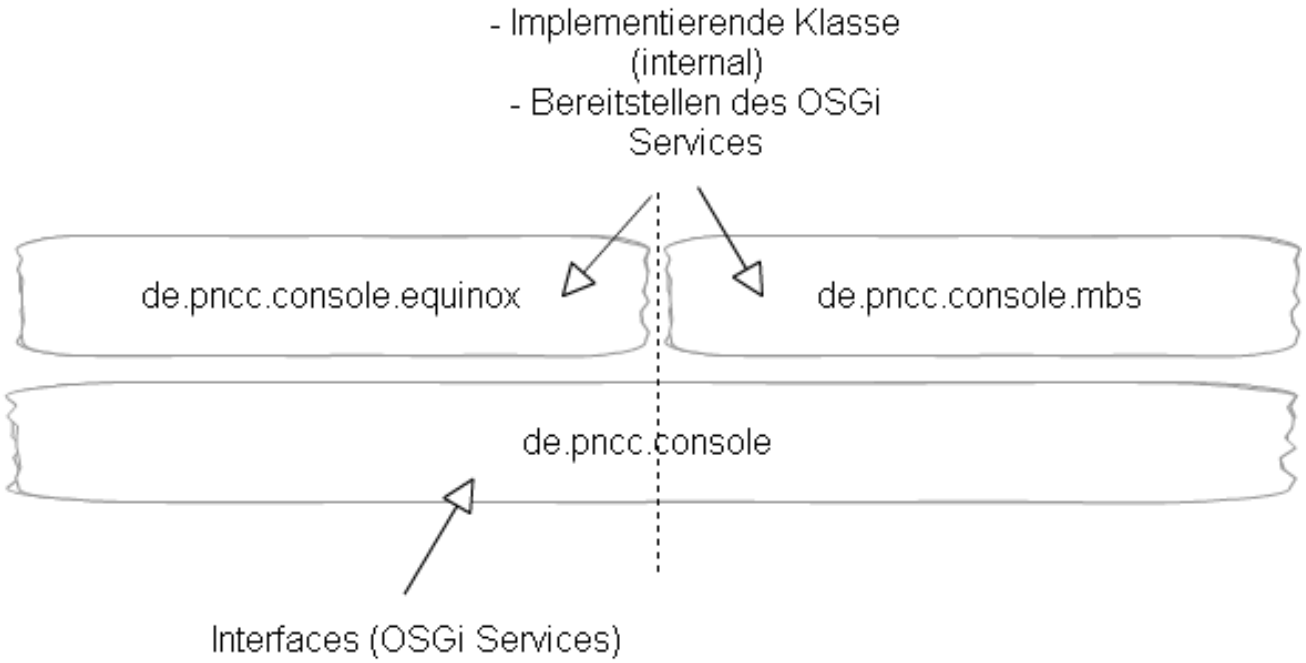
- Anforderungen
  - Speichern von Messungen
  - Speichern von konfigurierten Treibern und Transmittern

- Konzepte
  - NeoDatis ODB
    - Von ProSyst als angepasstes OSGi-Bundle bereitgestellt
    - GNU Lesser General Public License (LGPL)
  - Alle Messtypen als Object im Plug&Care Connector gehandelt
  - Konkrete Typen nur in Treiber- und Transmitter-Plugins
  - Satz an Standardtypen bereitgestellt (Kompatibilität)

Architektur Plug&Care Connector

Abstraktionen am Beispiel von Console

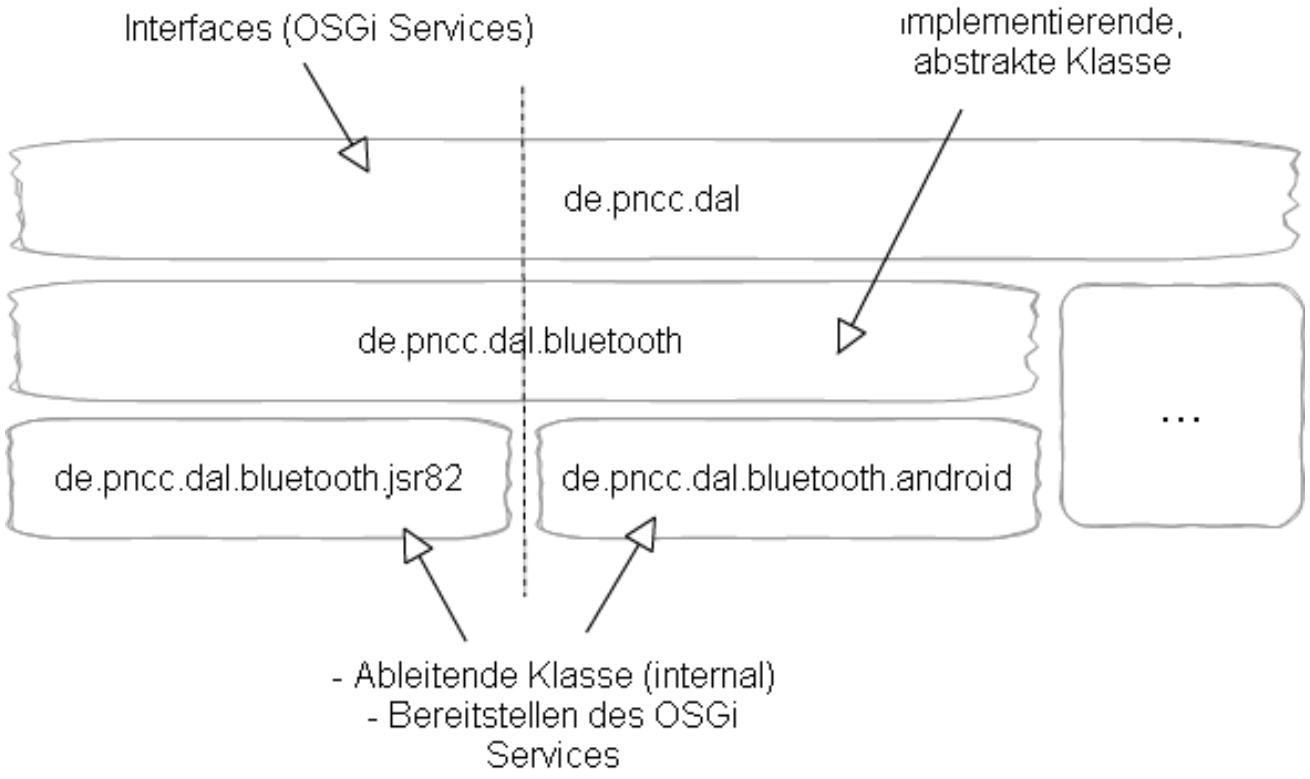
- Schichtenarchitektur beteiligter Bundles



Architektur Plug&Care Connector

Abstraktionen am Beispiel von Bluetooth

- Schichtenarchitektur beteiligter Bundles



Architektur Plug&Care Connector

Abstraktionen am Beispiel von Bluetooth

- DS Component Description von dal.bluetooth.jsr82

```
<component name=„de.pncc.dal.bluetooth.JSR82“>
    <implementation
    class="de.pncc.dal.bluetooth.jsr82.internal.JSR82BluetoothCommunicator"/>
    <service>
        <provide interface="de.pncc.dal.Communicator"/>
    </service>
</component>
```

Architektur Plug&Care Connector

Abstraktionen am Beispiel von Bluetooth und Console

- Deployment

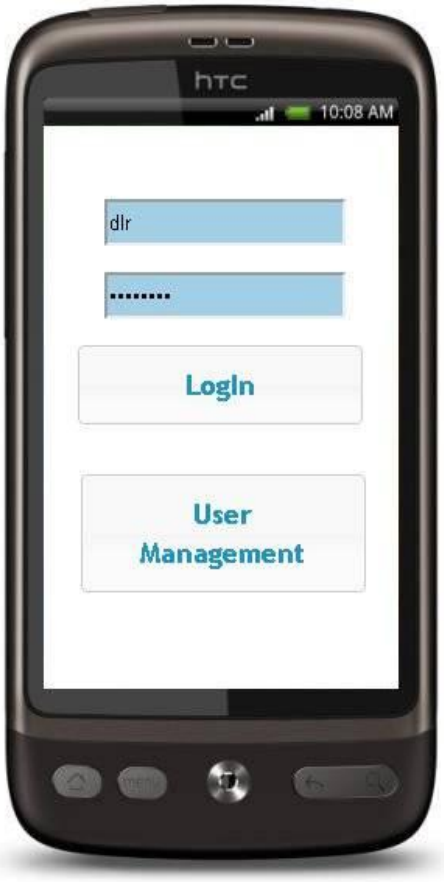
Architektur Plug&Care Connector

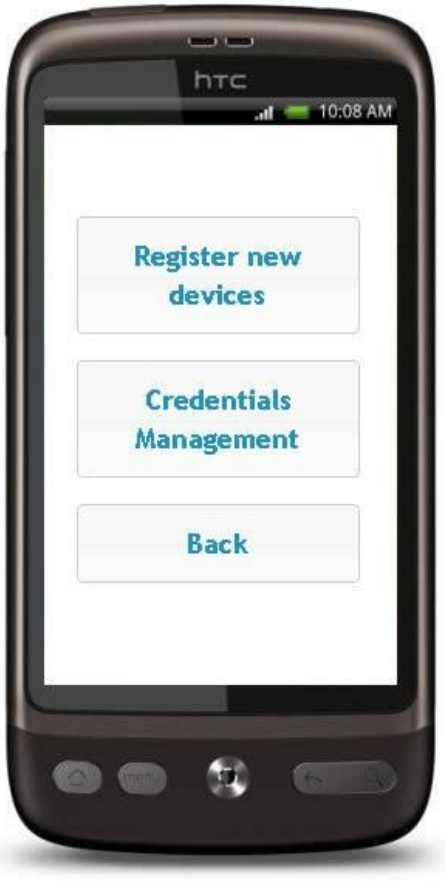
Third-Party-Libraries

- Plattform-unabhängig
  - Mitgeliefert in Bundle
  - Eventuell Probleme beim Classloading bei Android
- Plattform-abhängig
  - Als eigenständige Bundles bereitgestellt
  - Exportieren der gleichen Packages
  - Deployen jeweils eines Bundles pro Plattform

Plug & Care Connector

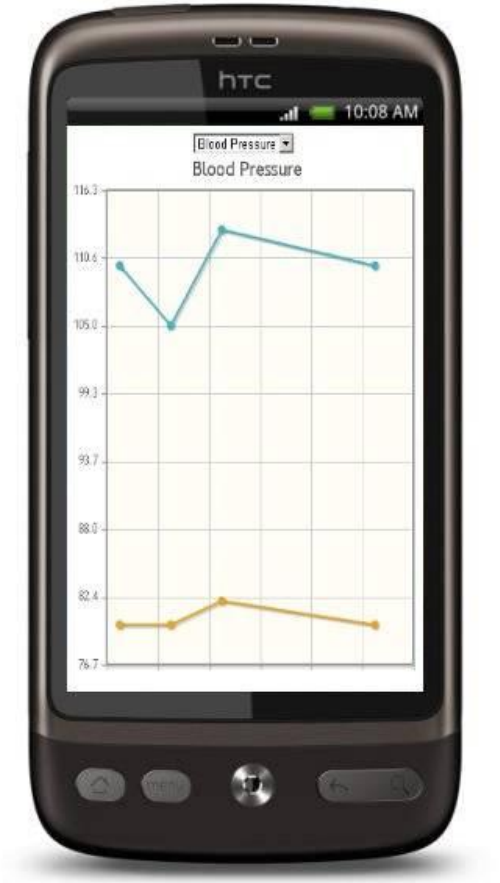
Screenshots: Settings Widget





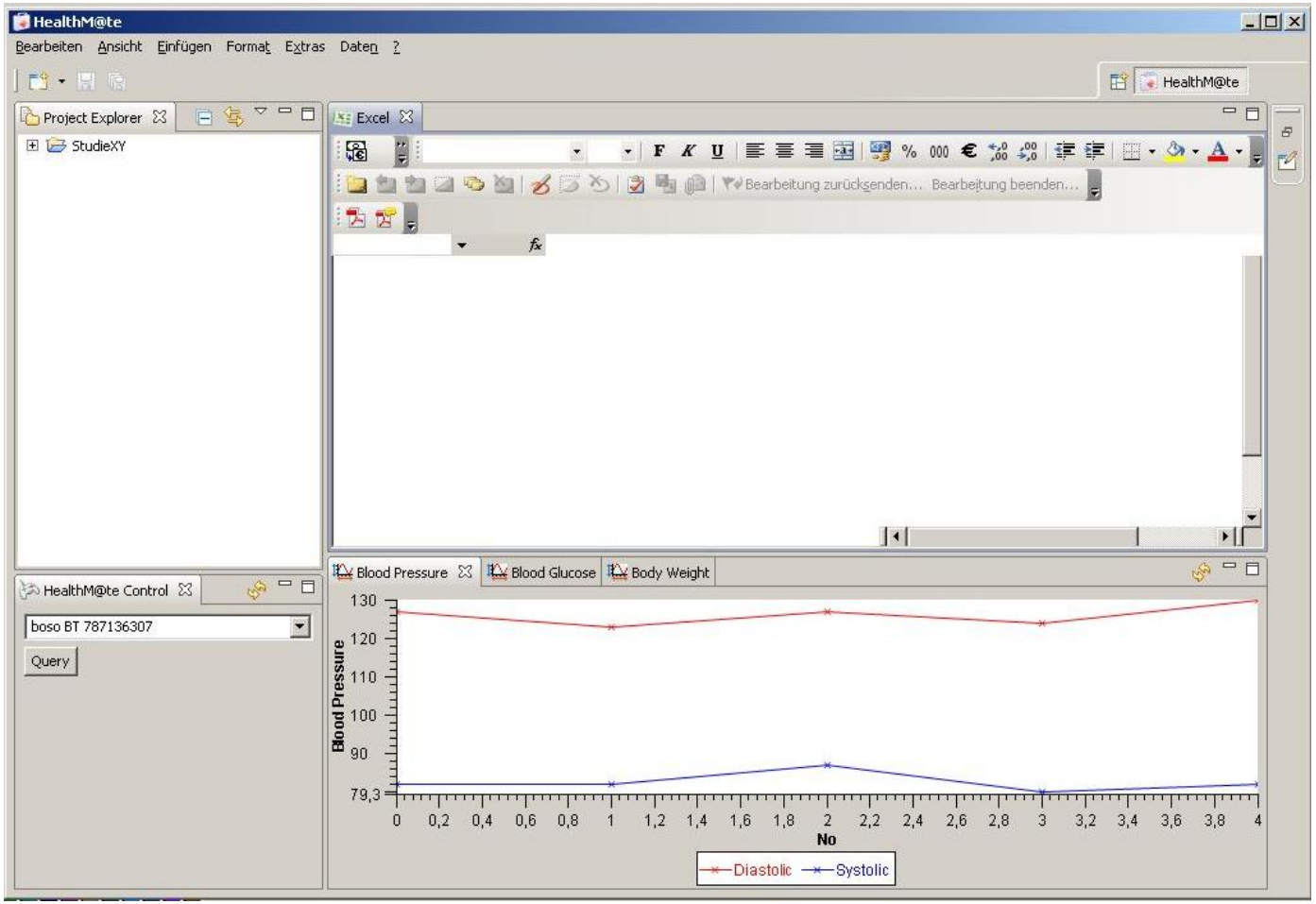
Plug & Care Connector

Screenshots: Chart Widget



Plug & Care Connector

Screenshots: RCP GUI



Best-Practices

- OSGi-Konzepte „ausleben“
  - Modularität
  - Lose Kopplung

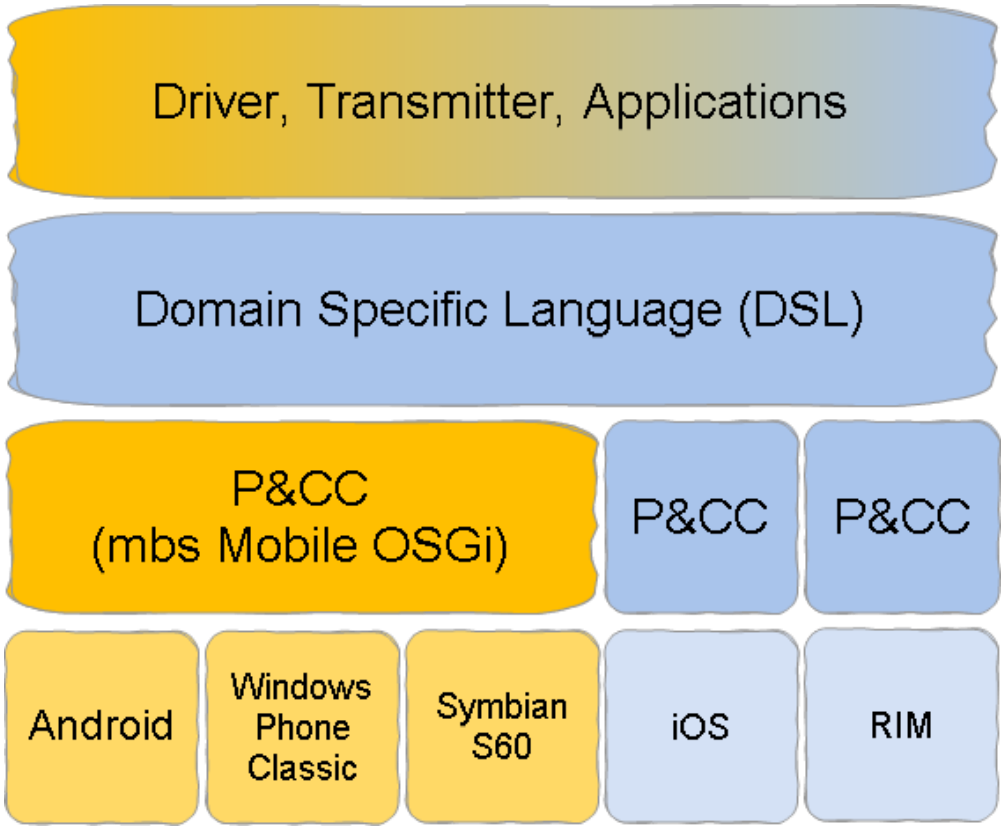
- Beispiele
  - Funktionalität von Bundles als OSGi-Services bereitgestellt
  - Individuelles Bundle-Setup beim Deployen
  - DS Component Model für Transparenz da Drittentwickler vorhanden
  - Third-Party-Libraries als OSGi-Bundles zwecks individuellem Setups

Probleme

- Platformunabhängigkeit
  - „Classloader-Hölle“ ~~(Write once, run anywhere.)~~ → Testen!
  - Performanz von Widgets → Native GUIs
  - Abhängig von Weiterentwicklung der OSGi-Stacks (z.B. momentan keine Unterstützung für Windows Phone 7)

Ausblick

Architektur



Ausblick

Anwendungsgebiete

- Sportmedizinischer Sektor
  - Überwachung von Puls, etc.
  - Kein zusätzliches Gerät zur Aufzeichnung durch (steigende) Verbreitung des Smartphones

- Experiment aus der Raumfahrt
  - Neutralisieren von Harn
  - Überwachung des Experiments
  - Durchführung des Experiments nach Model SETI@Home

Kontakt

Doreen Seider
Abteilung Verteilte Systeme und Komponentensoftware (SC-VK)
DLR Simulations- und Softwaretechnik
Köln-Porz / Braunschweig / Berlin
E-Mail: Doreen.Seider@dlr.de
www.dlr.de/sc/abteilung/verteiltesysteme