C++ Microservices Pt. II

Das QmitkServiceListWidget

Was bisher geschah

- Einführung in Microservices
- Spezielles Interface wird von Services deklariert
- Ab dann kann man sie MITK/MBI-weit nutzen

Was bisher geschah

- Ein Service hat:
  - Eine Klasse
    (z.B. mitk::USDevice)
  - Einen Interfacenamen
    (z.B. „org.mitk.services.UltrasoundDevice “)
  - Properties: Hashmap mit Eigenschaften
    (z.B. Properties[„SerialID“] = „a63d9fvd98“)

Microservices sind implementiert

- Jetzt soll der Benutzer mit ihnen interagieren
- Anhand typischer Tasks werden drei schon bestehende Fälle in MITK geschildert
  1) Services auflisten
  2) Reagieren auf Events
  3) Services einander zuordnen

QmitkServiceListWidget

- Ist der One-Stop-Shop!
- Abstrahiert Interaktion mit MicroServices
- Dokumentation: http://docs.mitk.org/nightly-qt4/classQmitkServiceListWidget.html
- Sieht so aus:

QmitkServiceListWidget

- Ist der One-Stop-Shop!
- Abstrahiert Interaktion mit MicroServices
- Dokumentation: http://docs.mitk.org/nightly-qt4/classQmitkServiceListWidget.html
- Sieht so aus:

Fall 1: Services auflisten

- Ultraschallmodul: Verschiedene Geräte können angelegt werden
- Gerät ist Connected = true/false
  - (true: kann jederzeit aktiviert werden)
- Und Active = true/false
  - (true: liefert derzeit Bilddaten)

Fall 1: Services auflisten

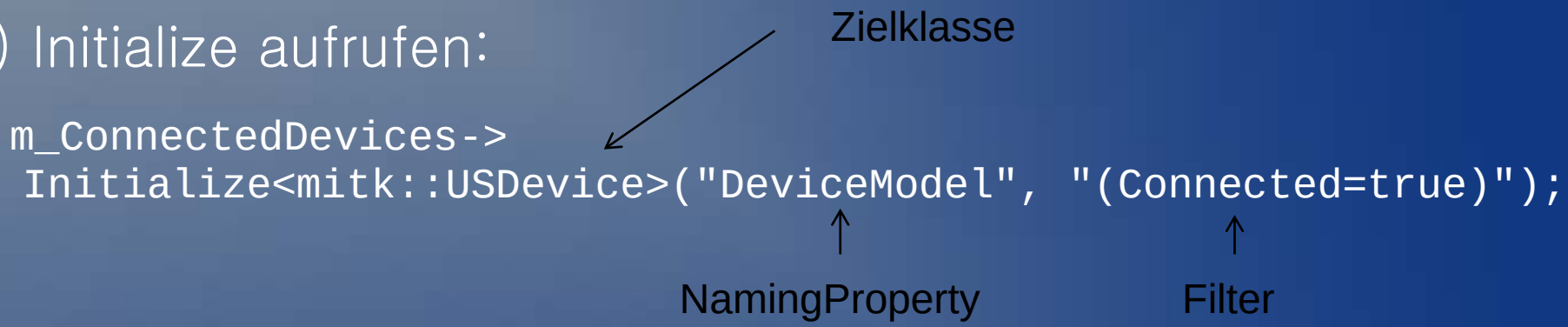
Nutzer soll aus den Connected Devices eines auswählen können um es zu aktivieren

Fall 1: Services auflisten

1) QmitkServiceListWidget in Plugin einfügen

   Name: `m_ConnectedDevices`

2) Initialize aufrufen:



3) Ausgewählten Service holen

```
mitk::USDevice::Pointer device =
GetSelectedServiceAsClass<mitk::USDevice>();
```

Ergebnis:

Esaote MyLab 5
Ultrasonix Pro
Superound 2000

Fall 2: Auf Events reagieren

- Für eine Anwendung muss ein Ultraschallgerät und ein Tracking Gerät ausgewählt sein
- Button „auswählen“ soll nur aktiv sein, wenn zwei Geräte angewählt sind

Fall 2: Auf Events reagieren

- ListWidget bietet Signale für Ereignisse

```
connect( m_Controls.m_USDevices, SIGNAL(
  ServiceSelectionChanged(mitk::ServiceReference) ), this,
  SLOT(OnClickDevices()) );

 connect( m_Controls.m_TrackingDevices, SIGNAL(
  ServiceSelectionChanged(mitk::ServiceReference) ), this,
  SLOT(OnClickDevices()) );
```

Fall 2: Auf Events reagieren

- Signale Verarbeiten:

```
if ( (m_Controls.m_USDevices->GetIsServiceSelected()) &&
 (m_Controls.m_TrackingDevices->GetIsServiceSelected()) )

   m_Controls.m_BtnSelectDevices->setEnabled(true);

 else

   m_Controls.m_BtnSelectDevices->setEnabled(false);
```

Signals:

- ServiceSelectionChanged: User wählt Service aus Liste aus
- ServiceRegistered: Neuer Service erschaffen
- ServiceModified: Service aus Liste modifiziert ODER bisher ungelisteter Service modifiziert, so dass er nun zur Liste passt
- ServiceModifiedEndMatch: Gelisteter Service Modifiziert, passt nichtmehr
- ServiceUnregistering: Service wird gleich abgemeldet

Fall 3) Komplexe Interaktion

Signale + Filter = Komplexe Interaktion möglich!

Beispiel aus IGT: Trackingeräte haben:

- eine NavigationDataSource
- eine ToolStorage

Beides sind Services, beide gehören zusammen

Fall 3) Komplexe Interaktion

Vorbereitung:

- DataSources bekommen property „UID“
- ToolStorages bekommen property „DataSource“
- Zwei ServiceListWidgets erstellen:
  - DataSources
  - ToolStorages

Fall 3) Komplexe Interaktion

Ablauf (Ohne Code)

- Reagieren auf: DataSources->ServiceSelectionChanged()
  - Property UID aus den Properties abfragen
  - Filter in ToolStorages ändern: DataSource = UID
- ToolStorages zeigt jetzt nur noch passende Geräte an

Fertig!

QmitkServiceListWidget:
Sieht nicht nach viel aus, kann aber einiges =)