Entwicklercamp 2009

Integration von Swing-basierten Java-Anwendungen in Notes 8

Karsten Lehmann, Tammo Riedinger

Geschäftsführer, Mindoo GmbH

Agenda

- Motivation
  - Vorteile und Nachteile der Integration in Lotus Notes 8
- Integrationsszenarien
  - Integration in Benutzerinterface
  - Datenintegration
  - Interaktion mit anderen Komponenten
- Beispielszenario
- Schritt für Schritt
  - Eclipse-Plugin erstellen
  - Toolbars und Menüs definieren
  - Erweiterung zu Composite Application-Komponente

Motivation

- Der neue Lotus Notes-Client ermöglicht die Integration bestehender Anwendungen in einer gemeinsamen Anwendungsoberfläche
- Dies kann die zentrale Pflege der Anwendungen vereinfachen
- Zudem bietet der Notes-Client eine Infrastruktur, mit der solche bestehenden Anwendungen erweitert werden und miteinander interagieren können
- Dieser Vortrag konzentriert sich auf die Integration bestehender Java-Anwendungen in Lotus Notes R8

Vorteile und Nachteile der Integration in Lotus Notes 8

- Vorteile:
  - Zentrale, einheitliche Anwendungsinstallation und Updatefähigkeit
  - Sicherheit in der Verteilung und Zugriffssteuerung
  - Erweiterung des Notes-Benutzerinterfaces mittels Standard-Eclipse-Extension Points
  - Größere Akzeptanz bei Nutzern durch vereinheitlichte Oberfläche
  - Direkter Daten- und Nachrichtentransfer zum Client
    - und dies sogar zu anderen Komponenten von Drittherstellern
  - bei Anwendungen mit Notes-API-Zugriff:
    Vermeidung von Konfigurationsproblemen aufgrund unterschiedlicher Lotus Notes-Installationsvarianten
- Nachteile:
  - Single Point of Failure
  - Flexibilität bei der parallelen Nutzung von Anwendungen ist geringer (Aufteilung auf virtuelle Desktops)
  - Erhöhter Speicherverbrauch gegenüber Lotus Notes Basic Client

Durch diesen Vortrag dargestellter Ansatz

- Entwicklung von Eclipse-Plugins, aber Wiederverwendung von bestehendem Code
- Häufig ist bereits Code vorhanden; Reimplementierung zu aufwändig/teuer
- schrittweise Umwandlung ist erwünscht

Konzepte zur visuellen Integration: Re-Parenting

- Fenster Re-Parenting
  - Beispiel: alter Notes-Client innerhalb des Lotus Notes Standard Clients R8
  - Das komplette Benutzerinterface wird ohne Anpassungen in den Client eingebettet
  - Kein Zugriff auf Sourcen/keine Änderungen an bestehender Anwendung nötig
  - Interaktionsmöglichkeiten sind beschränkt und abhängig von den Eigenschaften der bestehender Anwendung
  - Applikationen können zur Open-List vom Client hinzugefügt und normal von dort aus gestartet werden
  - Ansatz funktioniert für so gut wie jede Anwendung (in der Windows-Welt)
    - Applikationen, die mehrere top-level Fenster öffnen sind problematisch
    - Ebenso sind Applikationen problematisch, die schlecht auf Größenänderung ihrer Fenster reagieren

Technologie: Fenster Re-Parenting

DEMO

Konzepte zur visuellen Integration: OLE/COM

- OLE/COM-Embedding
  - Beispiel:
    Excel/Word/Internet Explorer einbetten
  - OLE-Embedding von ActiveX-Controls wird von Eclipse unterstützt
  - Integration:
    Direkte Ansteuerung von Funktionen und Datenaustausch, Abfangen von Events (z.B. Anklicken von Elementen im Browser)
  - Nachteile:
    - Windows only
    - Instabil z.B. bei Verwendung von Office-Komponenten

Technologie: OLE/COM

DEMO

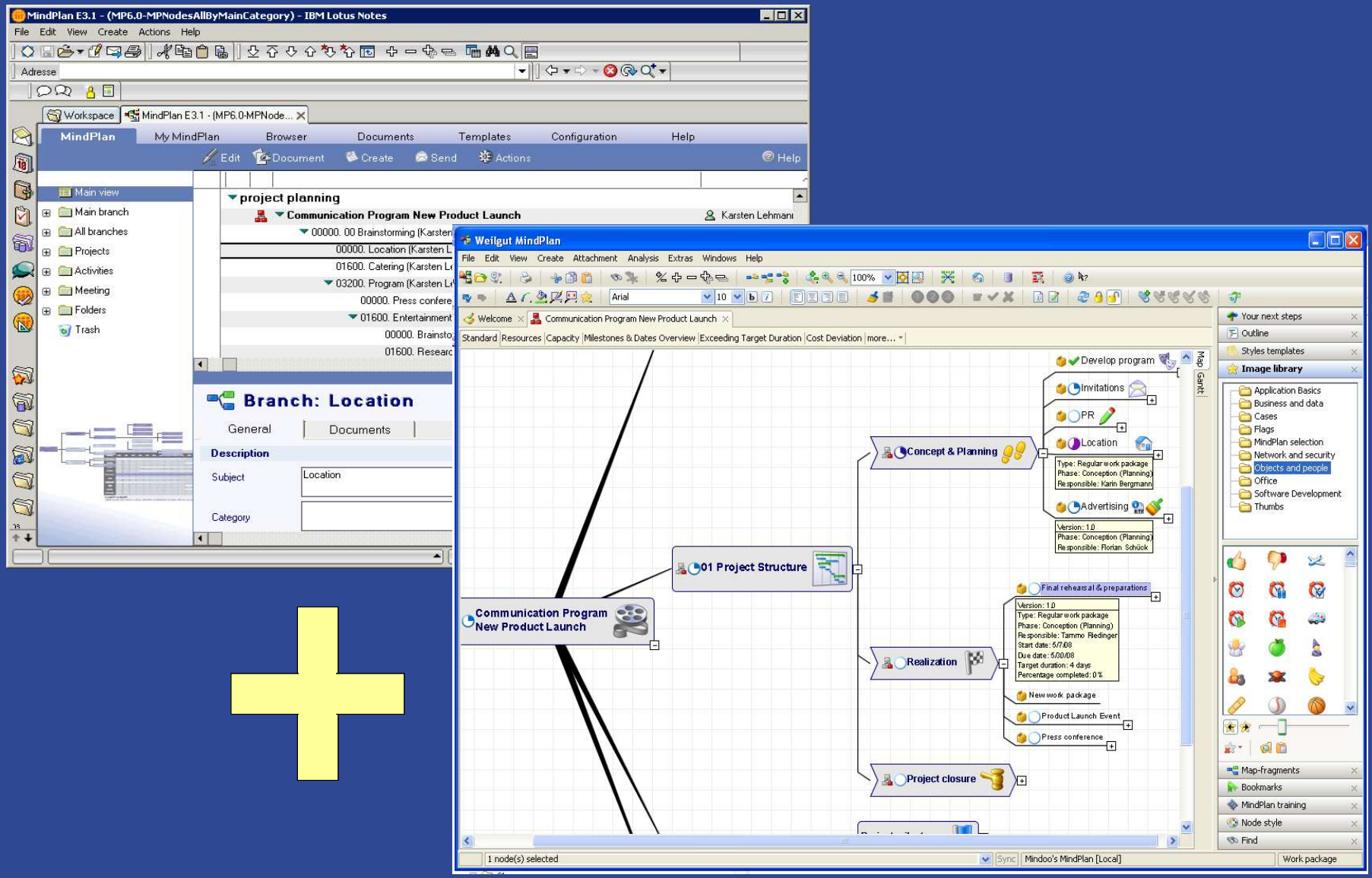
Konzepte zur visuellen Integration: Java-Elemente

- Einbindung vorhandener Java-Elemente
  - Hoher Integrationsgrad, weil gleiche Programmiersprache und Runtime (JVM)
  - Vorteile:
    - Direkter Datenaustausch, direkte Funktionsaufrufe
    - Änderung durch Klassenvererbung
  - Ausprägungen:
    - Re-Parenting (komplettes Java-Fenster inkl. Menüleiste einbetten)
    - Komponenten (z.B. HTML-Editor)
    - Eclipse-Plugin (standardisierter Weg, direkte Nutzung der Eclipse-Infrastruktur, wiederum erweiterbar durch fremde Plugins)

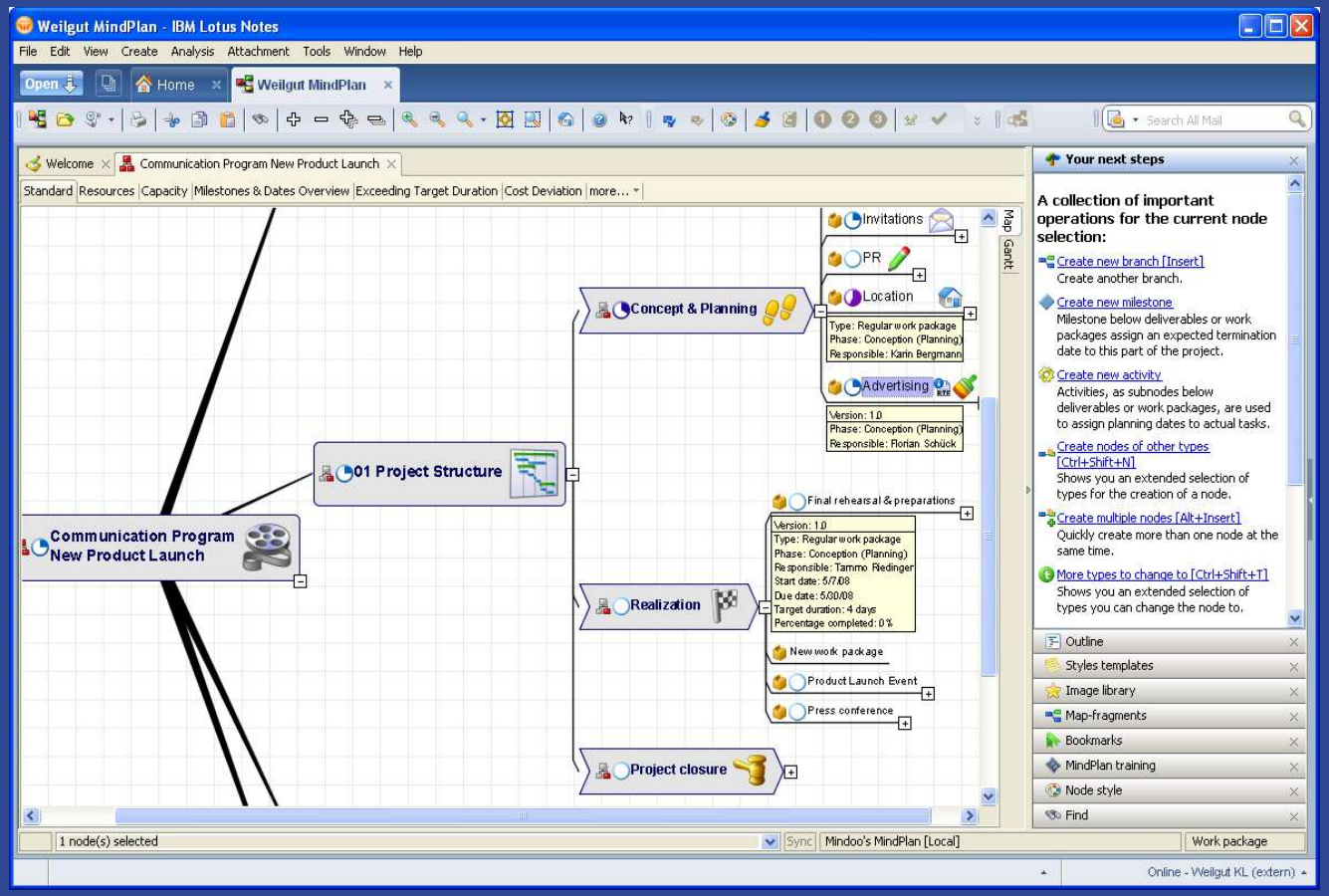
Unsere Aufgabe

- Eigenständige Swing-Applikation liegt vor und soll schrittweise in den Notes-Client integriert werden
- Das Vorgehen wurde u.a. bereits von uns umgesetzt bei der Software MindPlan für unseren Partner Haus Weilgut

Integrationsbeispiel: MindPlan



Integrationsbeispiel: MindPlan



MindPlan live

DEMO

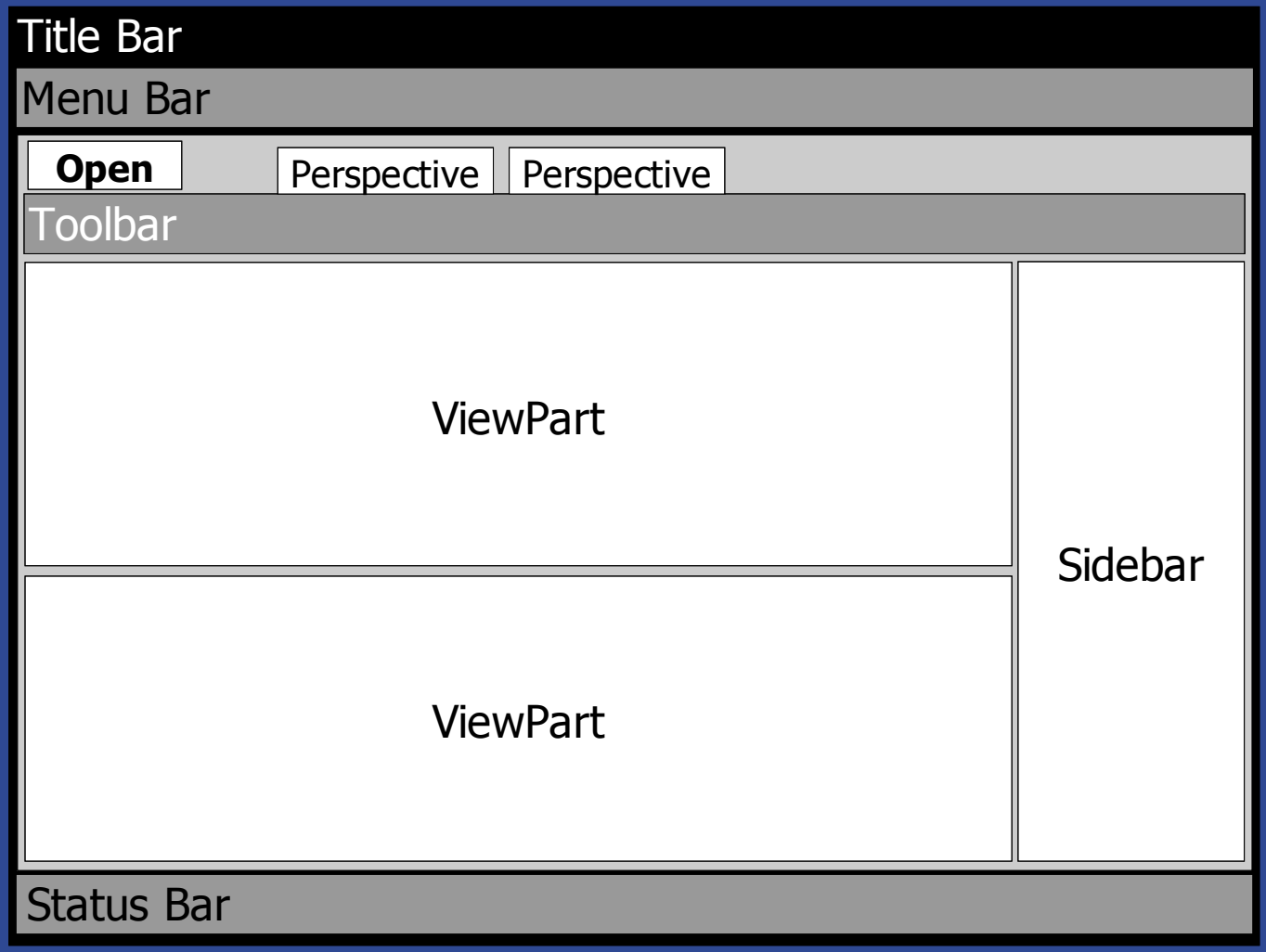
Motivation

- Wie haben wir dies erreicht?
  - Lesen, lesen, lesen – über Eclipse und Expeditor
  - Mit großem Durchhaltevermögen und natürlich viel Kaffee!
- Wie können Sie dies schaffen?
  - Wir versuchen Ihnen mit einer Sammlung nützlicher Dinge zu helfen, die wir Ihnen zu dieser Präsentation zur Verfügung stellen
    - Auflistung einiger wichtiger Eclipse/Expeditor Extension Points
    - Quellcode, der einige Dinge im Detail zeigt (anhand von einfachen Beispielen)
    - Link-Liste zu nützlicher Dokumentation und Einstiegspunkten
  - Wahrscheinlich ebenso mit viel Kaffee!

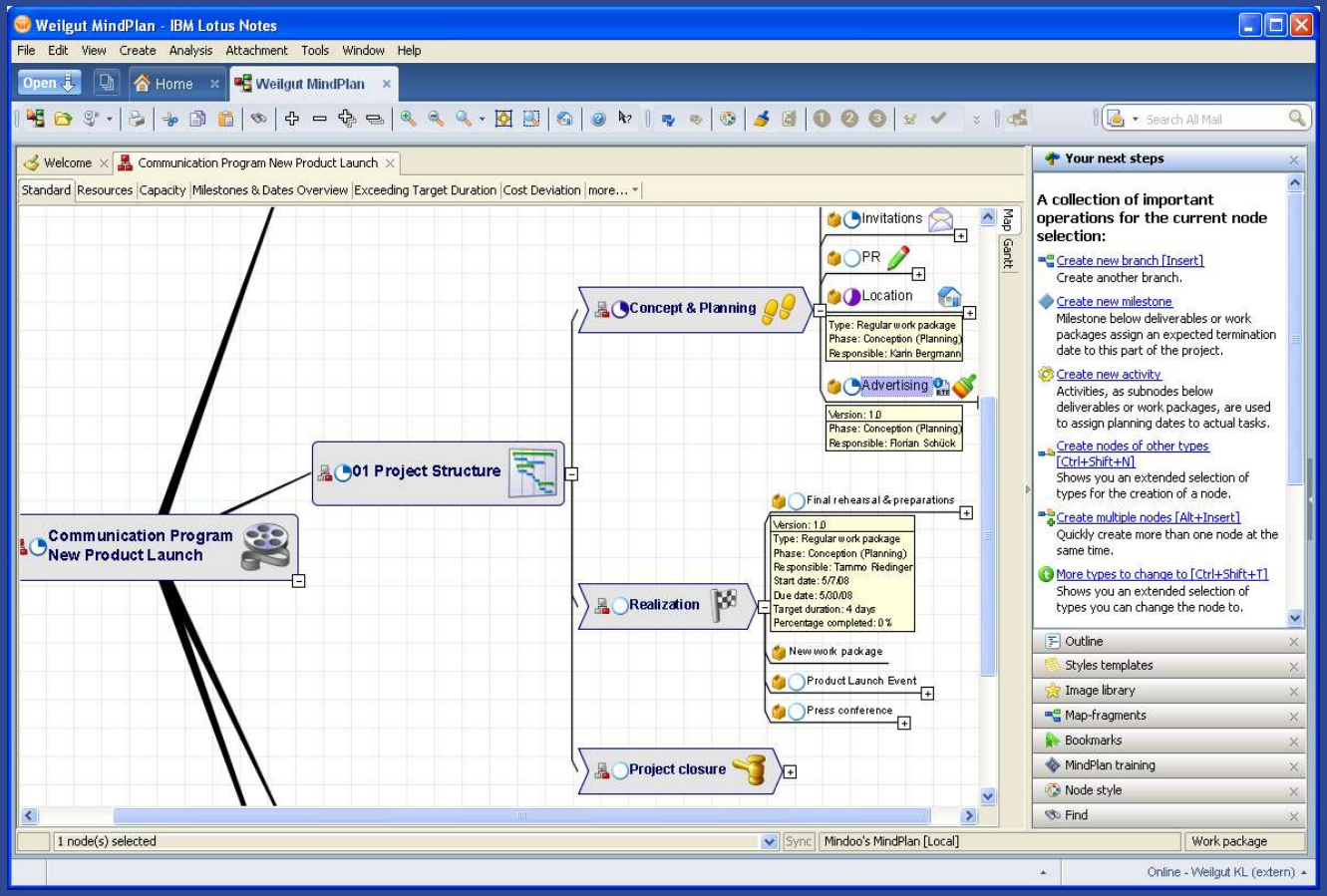
Integration – UI (1)

- Perspective
- ViewPart
- Menüs/Toolbars
- Extension Points erweitern und anbieten, z.B.
  - Eintrag in Open-Liste vom Lotus Notes Client
  - Aktionen zu Lotus Notes Java Views ergänzen
  - Kontextmenü der Mail-Datenbank erweitern
  - Elemente in der Lotus Notes Client Sidebar und vieles mehr (siehe Eclipse- und Expeditor-Dokumentation)
  - Livetext vom Lotus Notes Client unterstützen
  - Neue Extension Points für Plugins von Drittanbietern einführen!

Integration – UI (2)



Integration – UI (3)



Menüzeile / Open-Liste

Toolbars

Viewpart in Perspective

Integration – Swing/SWT

- SWT/AWT-Brücke, um Swing-Komponenten in SWT-UI nutzen zu können (Vorsicht Falle: zwei unterschiedliche Event-Threads!)
- Beispiel: Swing-Aufrufe aus SWT-Thread

```
// safely calling Swing code from SWT thread (will execute in Swing EDT)
Runnable r=...

// invoke on Swing Event-thread, asynchronously / synchronously
SwingUtilities.invokeLater(r); / SwingUtilities.invokeAndWait(r);

// check for Swing event-thread
if (SwingUtilities.isEventDispatchThread())
        r.run();
else
        SwingUtilities.invokeLater(r);
```

Integration – Swing/SWT

- Beispiel: SWT-Aufrufe aus Swing-Thread

```
// safely calling SWT code from Swing thread (will execute in SWT EDT)
Runnable r=...

// invoke on SWT Event-thread, asynchronously / synchronously
Display.getDefault().asyncExec(r); / Display.getDefault().syncExec(r);

// check for Swing event-thread
if (display.getThread() == Thread.currentThread())
        r.run();
else
        display.asyncExec(r);
```

Integration – Swing/SWT

- Nutzung des Eclipse Albireo-Projektes empfohlen
  - Setzt auf SWT/AWT-Brücke auf und ist leicht zu nutzen
  - Schafft Abhilfe für diverse Probleme
    - Modale Swing-Dialoge nicht modal zu SWT-Fenstern
      (z.B. könnte Lotus Notes Client-Fenster in den Vordergrund geholt werden, obwohl ein modaler Swing-Dialog angezeigt wird)
    - Verbessert die Fokus-Behandlung insgesamt
    - Vermindert Flackern der Swing-Komponenten beim Verändern der Größe
    - Ermöglicht es, SWT Popup-Menüs auf Swing-Komponenten anzuzeigen

Integration – Swing/SWT

- Einfaches Beispiel für Nutzung des Albireo-Projektes

```
// create an SWT-control that displays a Swing-Component
Control c=new SwingControl(parent, SWT.NONE) {
        protected JComponent createSwingComponent() {
                return new JButton("Press to see me swinging!");
        }

        public Composite getLayoutAncestor() {
                return parent;
        }
};
```

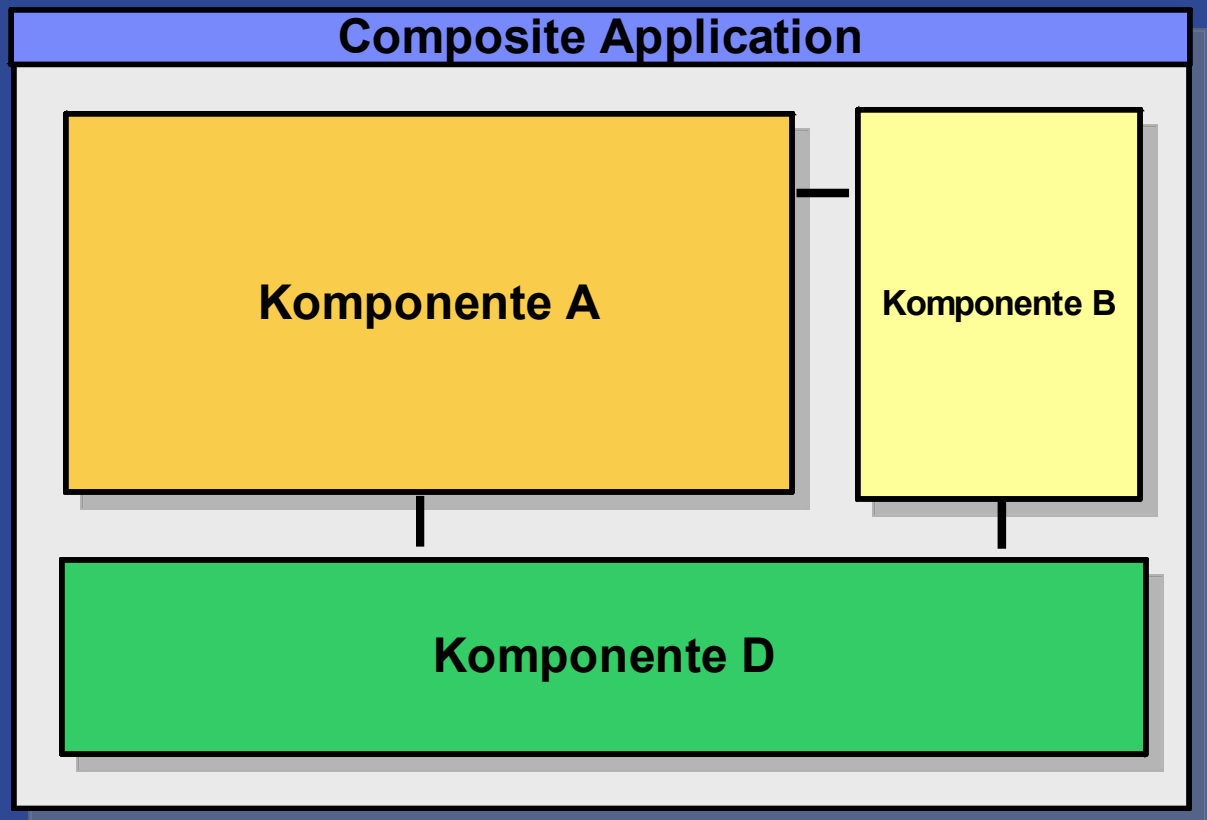
Swing-Komponente in der SWT-Welt

DEMO

Integration – Applikationsdaten

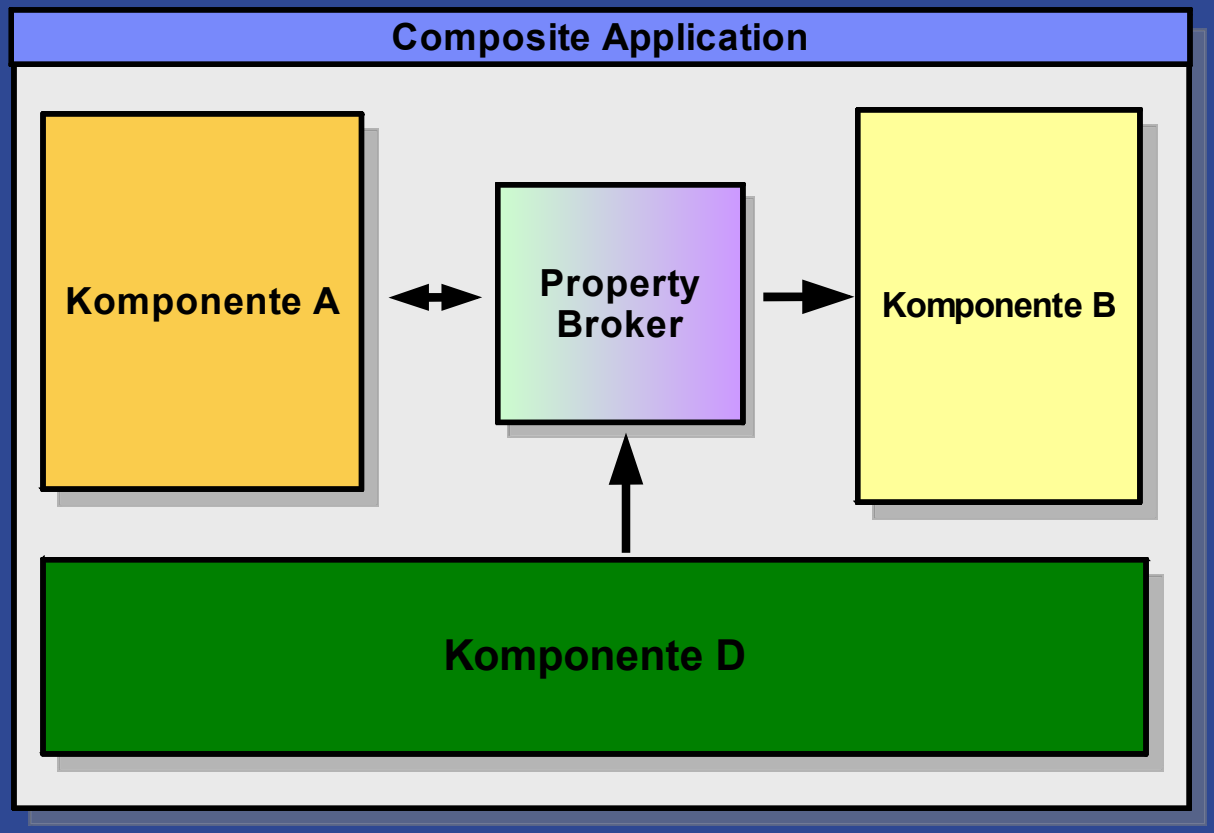
- Einfachere Möglichkeit auf Lotus Notes-Daten zuzugreifen
  - Keine komplizierte externe Installationsroutine nötig (kann bisweilen beim Aufspielen ein wahrer Alptraum sein)
- Verbindung mit Eclipse-Diensten
  - Interne Selektionen für Eclipse verfügbar machen (z.B. Text, URIs zu Notes Dokumenten)
  - Auf externe Selektionen reagieren (z.B. von Lotus Notes)
- Property Broker für Kommunikation zwischen unterschiedlichen Komponenten
  -> Composite Application

Composite Application – das Konzept



- **Zusammenstellung** unterschiedlicher **Komponenten**
- Nutzung unterschiedlicher Technologien
- Spezifisch für Business-**Kontext**
- Visuelle Integration
- **Verbunden** für den Informationsaustausch

Composite Application – das Konzept

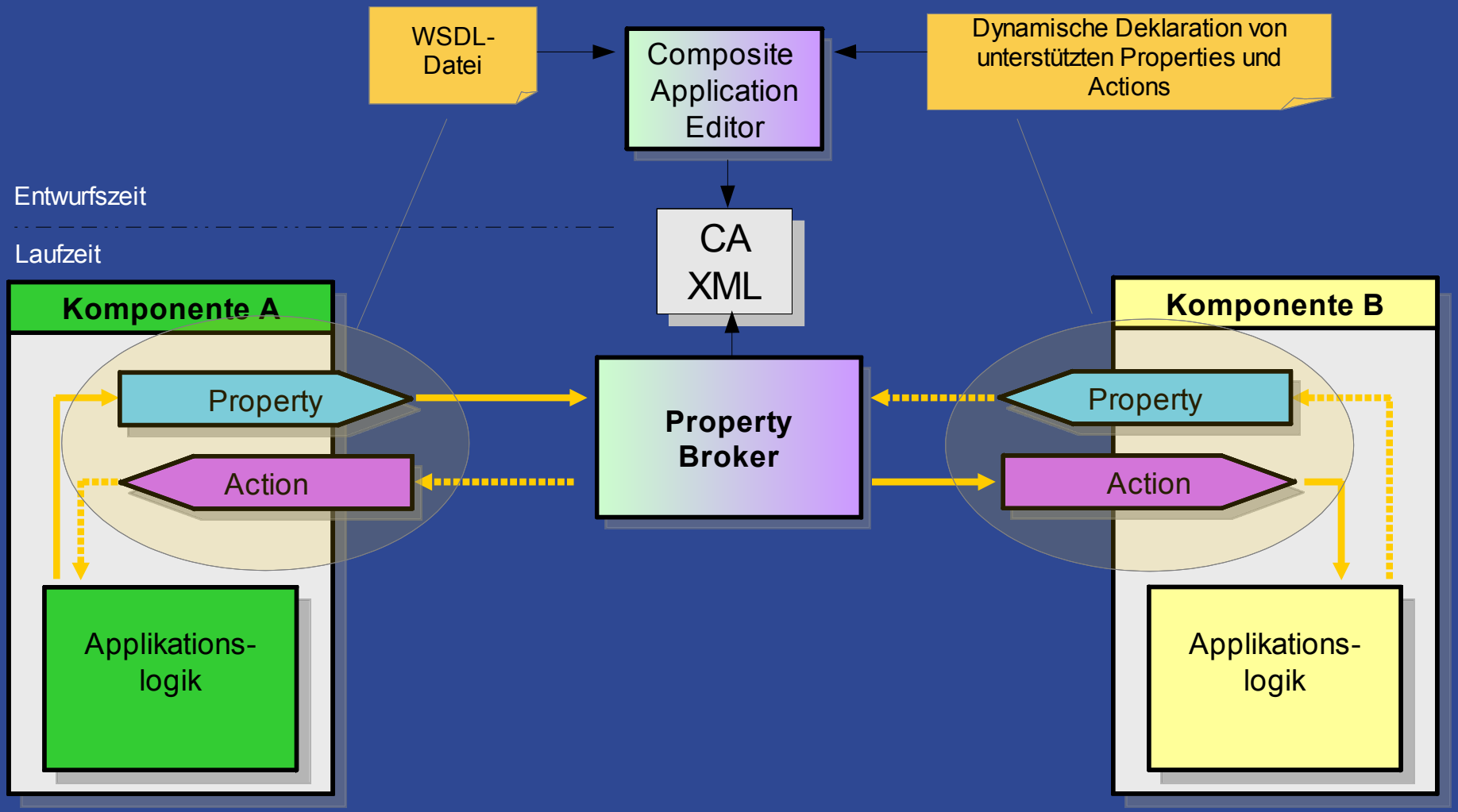


Die Kommunikation zwischen den Komponenten erfolgt über den Property Broker.

Kommunikation mittels Property Broker

- Komponenten-Interaktion
  - **Lose verknüpft** durch Eigenschaften (property) und Aktionen (action)
  - **mix-and-match** von Komponententechnologien
  - **Property** ist ein typisiertes, übertragbares Datum
- **Action** ist die Logik, um eine Eigenschaft entgegenzunehmen
- Komponenten definieren ihre Eigenschaften und Aktionen selbst
- **Wires** verbinden Eigenschaften mit Aktionen
  - 1 zu 1 oder 1 zu n Verbindungsoptionen

Kommunikation mittels Property Broker



Verfügbare Properties und Actions werden entweder durch WSDL-Dateien bekannt gemacht oder können dynamisch zur Laufzeit deklariert werden.

Beispiel für eine Composite Application

DEMO

Schritt für Schritt – unser Beispiel

DEMO

Schritt für Schritt – die Umsetzung

- Drei Schritte zum Erfolg :-)
  - Eclipse Plugin-Projekt erstellen
  - Toolbar-Aktionen und Menüpunkte definieren und mappen
  - In Composite Application umwandeln!

Schritt für Schritt – Eclipse Plugin-Projekt

- Enthält alle Ressourcen, aus denen Ihre vorhandene Applikation besteht
- Definiert Erweiterungen zum Lotus Notes Standard Client per plugin.xml
- Ein Applikationsreiter entspricht einer Eclipse Perspective
  - Eine Perspective definiert das Layout der ViewParts, welche die Applikations-Daten anzeigen
  - Mögliche Abbildung: eine Perspective mit einem ViewPart
    - Einfachster Ansatz, eine komplette bestehende Anwendung einzubetten
    - Es ist natürlich möglich, mehrere ViewParts (und perspectives) für z.B. Dokumenten-orientierte Anwendungen zu definieren

Schritt für Schritt – Applikations-Lebenszyklus

- START: Applikation initialisieren (z.B. Bei der Erstellung des ersten ViewParts)
  - Einbettung von Swing-Komponenten der Applikation in ViewParts
  - Es empfiehlt sich, eine Applikation zu trennen in ein Applikationsobjekt und Nutzerinterface-Komponenten
- DISPOSE: Swing-Komponenten verwerfen, wenn ein ViewPart geschlossen wird
  - Möglichst versuchen, Speicher freizugeben (viele plugins teilen sich diesen!)
- STOP: Applikation herunterfahren, wenn letzter ViewPart geschlossen wird
  - Achtung: System.exit() ersetzen, da dies sonst Lotus Notes beenden würde!
  - Singletons und statische Member-Variablen in der alt-Anwendung sind problematisch.
    Diese müssen verworfen werden, wenn die Applikation beendet wird. Andernfalls existiert kein sauberer Zustand, um eine neue Applikationsinstanz starten zu können.

Schritt für Schritt – Toolbars und Menüs (1)

- Action sets/control sets
  - Definition von Aktionen und zugehörigen Toolbar- und Menü-Elementen
  - Control Set ist eine Erweiterung zu action sets vom Expeditor-Toolkit
    - Einfügen von Komponenten auf Toolbars (Combo-Boxen o.ä.)
    - Beide Varianten können im Plugin gleichzeitig genutzt werden
  - Statische Definition
    - Definition per plugin.xml zur Entwurfszeit
    - Sichtbarkeit und Aktivierung definierbar über Muster (z.B. In Abhängigkeit zu Selektionen)
  - Dynamische Definition
    - Action Sets/Control Sets zur Laufzeit hinzufügen
    - Dieser Ansatz sollte für bestehende Anwendungen flexibler sein
    - Erhöht die Entkopplung von Alt-Anwendung und integrierter Anwendung, so dass unter Umständen beide mit der gleichen Code-Basis weiterexistieren können
    - Aktionen nur einmal definieren und diese Definition von einer Brücken-Komponente auslesen
  - Action Sets/Control Sets können auch per Code aktiviert oder deaktiviert werden
    - Um damit Menü- und Toolbar-Elemente abhängig von internen Zuständen der Applikation anzuzeigen oder zu verstecken

Code-Beispiel verfügbar!

Schritt für Schritt – Toolbars und Menüs (2)

- Kontexte
  - Aktionen können zu mehreren Kontexten gehören
  - Aktivierung/Deaktivierung von Aktionen über Action Sets/Control Sets hinweg
- Commands und Bindings
  - Verbinden Aktionen mit Kontexten und Tastaturkürzeln

```
IContextService ctxService=(IContextService)getSite().getService(IContextService.class);

IContextActivation activeContext=ctxService.activateContext("contextID");
...
ctxService.deactivateContext(activeContext);
```

- Sonderrolle Popup-Menüs und Drop-down actions
  - Möglichkeit, vorhandene Aktionen direkt ohne Eclipse-Delegates zu nutzen
  - Empfehlung: So oft wie möglich nutzen, um vorhandenen Code dynamisch einbinden zu können

Code-Beispiel verfügbar!

Dynamische Menüs mit Action Sets und Control Sets

DEMO

Schritt für Schritt – Composite Application (1)

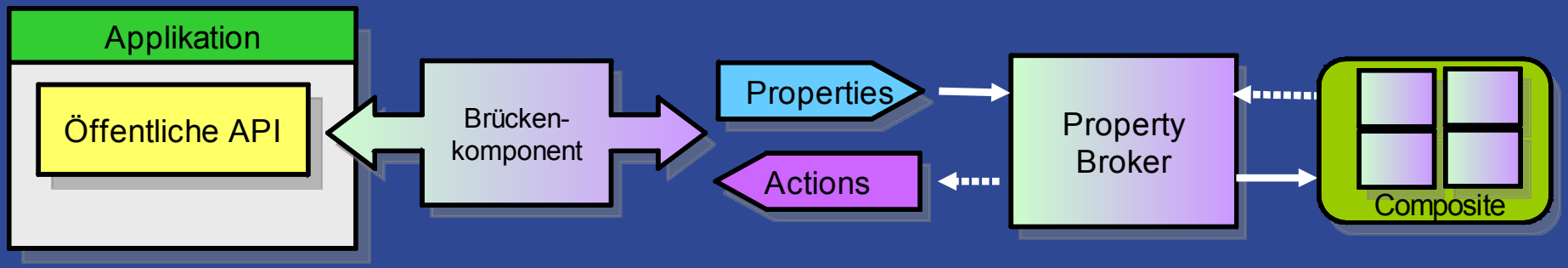
- Führe Properties und Actions ein, um eine Interaktion mit fremden Komponenten zu ermöglichen
- Identifiziere relevante Properties und Actions
  - Ein guter Ansatzpunkt ist es, interne Selektionen nach außen verfügbar zu machen
  - Beispiele nützlicher Eigenschaften
    - IDs interner Datenobjekte (Flugnummer, Personalnummer, ID eines Unternehmens im CRM System, Notes Dokument UNID)
    - Ressourcen-Namen (Ersteller, Author, Betriebsname)
    - Daten (Aktivitäten, Erstellung)
  - Beispiele nützlicher Aktionen
    - CRUD: Create, Read, Update, Delete (für interne Datenobjekte)
    - Seletion ändern
    - Fokussierung ändern
    - Weitere Objekte öffnen

Schritt für Schritt – Composite Application (2)

- Unterschiedliche Nutzungsszenarien identifizieren
  - Statische Properties, um die CA-Komponente zu konfigurieren
    - Solche Properties können im CA-Editor genutzt werden, um bestimmte Modi der Applikation zu setzen/zu aktivieren
    - Edit-/Read-only Modus für CA-Komponente in bestimmten Composite Applications
    - Navigation unterbinden/steuern
    - Initialen Status setzen, z.B. Home Page in einer Browser-Komponente
- Statische/dynamische Deklaration von Properties/Actions
  - Statisch: Properties definieren mit WSDL zur Entwurfszeit
  - Dynamisch: mittels Extension Points zur Laufzeit
  - Dynamische Deklaration ist flexibler
  - Erzeugung neuer Properties in Abhängigkeit zum Status der Applikation oder der Daten-Objekte wird bei dynamischer Deklaration möglich

Schritt für Schritt – Composite Application (3)

- Mit Hilfe einer "Brücke" kann eine lose, einfach zu verwaltende Kopplung geschaffen werden
- Existierende Applikationen können so angebunden werden, ohne für den Property-Broker entwickelt worden zu sein
- Applikationen muss nicht re-implementiert werden
- Mögliche Lösung: nutze Java-Bean-Implementierung
  - Wiederverwendbares Muster für Brücken-Komponente
  - Wird genutzt in Plugins der CA Component Library (www.openntf.org)
  - Liest statische, initiale Properties vom Expeditor Topology Handler
  - Nutzt set/get-Methoden der Bean für das Senden/Empfangen von Properties



Unser Beispiel als Composite Application-Komponente

DEMO

Zeit für Fragen

Vielen Dank für Ihre Aufmerksamkeit!

Wir stehen nun für Fragen zur Verfügung.

Backup

Code-Beispiel: Dynamische Menüs/Toolbars

- Control Sets dynamisch anzeigen/verstecken:

```
// show/hide an control-set (remember this is Expeditor-API)
import com.ibm.rcp.ui.controlset.ControlSetHelper;

IWorkBenchWindow window=...
ControlSetHelper controlSetHelper=ControlSetHelper.getInstance();

// show (set last boolean to <code>true</code> to tell the UI to refresh
controlSetHelper.setControlSetVisible(window, "controlSetID", true, false);
...
// hide
controlSetHelper.setControlSetVisible(window, "controlSetID", false, true);
```

- Action Sets dynamisch anzeigen/verstecken:

```
// show/hide an action-set
getSite().getPage().showActionSet("actionSetID");
...
getSite().getPage().hideActionSet("actionSetID");
```