**Projekthandbuch**

Eclipse RCP ist ein Standardframework für Geschäftsanwendungen. Mit der neusten Generation E4 wurde Eclipse RCP vollständig modernisiert. Anhand einer wichtigen RCP Applikation der SBB wird eine Migration auf Eclipse E4 exemplarisch durchgeführt und die dabei berücksichtigten Aspekte dargestellt.

|  |  |
| --- | --- |
| Autor: | Mike Rothenbühler |
| Version: | 0.2 |
| Status: | In Arbeit |
| Ablage: | https://github.com/MikeR13/MAS/blob/master/Deliverables/ |
| Institution: | Hochschule für Technik und Informatik Bern |
| Verteiler: | Brawand Ueli, Hoffmann Marc, Rothenbühler Mike |

**Versionkontrolle**

|  |  |  |  |
| --- | --- | --- | --- |
| **Datum** | **Version** | **Autor** | **Bemerkungen** |
| 03.06.2013 | 0.1 | MIRO | Erster Wurf |
| 07.06.2013 | 0.2 | MIRO | Aspekt 1, Mixing E3 / E4 |
| 17.06.2013 | 0.3 | MIRO | Aspekt 1, Mixing E3 / E4 |

Inhaltsverzeichnis

1. Einleitung 5

1.1. Zweck des Dokumentes 5

1.2. Problemstellung 5

1.3. Randbedingungen 5

1.4. Situationsanalyse 5

1.5. Erbrachte Vorleistung 5

2. Vorbereitung 6

2.1. Architektur Eclipse 3 vs 4? Oder eher im Aspekt Nr.1 6

3. Aspekt Iterationen 6

3.1. Beschreibung des Aspektes „Mixing E3 / E4“ 6

3.2. Diskussion der Eclipse RCP 4 Lösung 6

3.2.1. Vorteile 7

3.2.2. Vergleich mit Eclipse RCP 3 7

3.2.3. Einschränkungen und Risiken 8

3.2.4. Qualität und Testbarkeit im Vergleich zu Eclipse RCP 3 8

3.3. Konkretes Beispiel RCS 9

3.3.1. Definition Abnahmekriterien 9

3.3.2. Migration 9

3.3.3. Test 9

3.4. Beschreibung des Aspektes „Adapters / Dependency Injection“ 9

3.5. Diskussion der Eclipse RCP 4 Lösung 11

3.5.1. Vorteile 11

3.5.2. Vergleich mit Eclipse RCP 3 11

3.5.3. Einschränkungen und Risiken 11

3.5.4. Qualität und Testbarkeit im Vergleich zu Eclipse RCP 3 11

3.6. Konkretes Beispiel RCS 11

3.6.1. Definition Abnahmekriterien 11

3.6.2. Migration 11

3.6.3. Test 11

3.7. Beschreibung des Aspektes „Commands / Handler, Menus, Key Bindings“ 11

3.8. Diskussion der Eclipse RCP 4 Lösung 11

3.8.1. Vorteile 11

3.8.2. Vergleich mit Eclipse RCP 3 11

3.8.3. Einschränkungen und Risiken 11

3.8.4. Qualität und Testbarkeit im Vergleich zu Eclipse RCP 3 11

3.9. Konkretes Beispiel RCS 11

3.9.1. Definition Abnahmekriterien 11

3.9.2. Migration 11

3.9.3. Test 11

3.10. Beschreibung des Aspektes „Eigene Extension Points / Eigene Services“ 11

3.11. Diskussion der Eclipse RCP 4 Lösung 11

3.11.1. Vorteile 11

3.11.2. Vergleich mit Eclipse RCP 3 11

3.11.3. Einschränkungen und Risiken 11

3.11.4. Qualität und Testbarkeit im Vergleich zu Eclipse RCP 3 11

3.12. Konkretes Beispiel RCS 11

3.12.1. Definition Abnahmekriterien 12

3.12.2. Migration 12

3.12.3. Test 12

3.13. Beschreibung des Aspektes „Application Model vs. Advisors“ 12

3.14. Diskussion der Eclipse RCP 4 Lösung 12

3.14.1. Vorteile 12

3.14.2. Vergleich mit Eclipse RCP 3 12

3.14.3. Einschränkungen und Risiken 12

3.14.4. Qualität und Testbarkeit im Vergleich zu Eclipse RCP 3 12

3.15. Konkretes Beispiel RCS 12

3.15.1. Definition Abnahmekriterien 12

3.15.2. Migration 12

3.15.3. Test 12

3.16. 12

4. Reflexion 12

Einleitung

* 1. Zweck des Dokumentes

Dieses Projekthandbuch informiert die an „Migration von Eclipse 3.x nach Eclipse 4“ beteiligten Parteien über die getätigten Migrationen von den ausgewählten Aspekten. Im Folgenden werden die Begriffe Eclipse 3.x RCP und E3 sowie Eclipse 4.x RCP und E4 identisch behandelt. Es wird also immer die Rede von Eclipse RCP und nicht der IDE sein.

* 1. Problemstellung

Da sich mit der Version 4 einiges an Eclipse RCP geändert hat ist eine Migration nicht einfach so zu bewerkstelligen. Es gibt aus der Community (noch) nicht viele Berichte zu gelungenen Migrationen, geschweige denn eine Anleitung wie eine solche Migration erfolgreich durchgeführt werden kann.

Es sollen Erkenntnisse gewonnen werden, wie eine erfolgreiche Migration durchgeführt werden kann, ohne dass die bestehende Applikation in den Punkten

* Funktionalität
* Performance
* Stabilität
* Usability
* Look and Feel

negativ beeinflusst wird. Die Arbeit an der bestehenden Applikation soll auch während der Migrationszeit möglich sein. Hierfür müssen Lösungen erarbeitet werden.

Mit den Erfahrungen und Ergebnissen aus der Master Thesis soll eine Migration auch für grosse Projekte relativ einfach möglich sein.

* 1. Randbedingungen

Das Projekt wird im Rahmen der Semesterarbeit und der Diplomarbeit durchgeführt. Dafür gelten die an der HTI üblichen Bedingungen.

* 1. Situationsanalyse

Um die erarbeiteten Migrationsvorschläge in der Praxis zu verifizieren, wird die Migration des RCP-Clients exemplarisch durchgeführt.

Bei dem RCS Client handelt es sich um die grösste Eclipse RCP Applikation der SBB mit folgenden Kennzahlen:

* 70 Plugins
* über 10 Hauptfenster
* Dutzende von Dialogen
* 270‘000 Zeilen Code

RCS wird zur Disposition des Zugverkehrs auf dem gesamten Streckennetz der SBB verwendet.

* 1. Erbrachte Vorleistung

Das Knowhow über Eclipse RCP 4 wurde bereits vor Projektstart aufgebaut.

Vorbereitung

* 1. Architektur Eclipse 3 vs 4? Oder eher im Aspekt Nr.1

Aspekt Iterationen

* 1. Beschreibung des Aspektes „Mixing E3 / E4“

In dieser Iteration soll geprüft werden welche Migrationsmöglichkeiten überhaupt existieren. Können Eclipse RCP 3 und Eclipse RCP 4 Komponenten im selben Projekt gleichzeitig nebeneinander im Einsatz sein? Um einen Kontext zu schaffen werden die beiden Versionen 3.x und 4.x kurz vorgestellt und einander gegenübergestellt.

* 1. Diskussion der Eclipse RCP 4 Lösung



Abbildung Architektur E4

E4 läuft auf einer Java Virtual Machine (Fussnote), die Java Version sollte mindestens 6 sein.

Eclipse 4 RCP besteht aus den Komponenten:

|  |  |  |
| --- | --- | --- |
| **Komponente** | | **Beschreibung** |
| Eclipse 4.x Application Platform | Modeled UI | Das Application Model ist ein Konzept von E4. Die komplette Anwendung befindet sich auch als abstraktes Modell im Speicher. |
| CSS | Cascading Style Sheets erlaubt es Benutzeroberflächen optisch zu verändern ohne Code anzupassen. Es können Schriften, Farben und weitere Teile verändert werden. |
| Dependency Injection | Mit Dependency Injection werden Abhängigkeiten von Objekten zu anderen Objekten vom Container augelöst und gesetzt. Dadurch wird der Code des Objektes unabhängig von seiner Umgebung. Solche Klassen lassen sich um ein Vielfaches einfacher Unittesten als Objekte die sämtliche Referenzen selbst erzeugen. |
| Application Services | Services die von E4 zur Verfügung gestellt werden. Das sind zum Beispiel Services für das Logging, Eventhandling und Zugriffe auf das Application Model. |
| Equinox | OSGi Implementation von Eclipse. Die Equinox Runtime stellt das Framework, um modulare Eclipse Applikationen laufen zu lassen, zur Verfügung |
| EMF Core | Das Eclipse Modeling Framework ist ein quelloffenes Java-Framework zur automatisierten Erzeugung von Quelltext anhand von strukturierten Modellen (link to wiki) |
| SWT/ JFace | SWT ist die Standard User Interface Komponentenbibliothek von Eclipse. JFace stellt praktische APIs oberhalb von SWT zur Verfügung. |
| Workbench | | Im Prinzip ein Adapter der Aufrufe an die e4-Bundles weiterleitet. Die Kompatibilitätsschicht übersetzt API-Aufrufe aus E3 in die neue Welt, also E4. |
| JDT | | Die Java Development Tools sind eine Reihe von Plug-ins für die Entwicklungsumgebung Eclipse (link to wiki) |
| PDE | | Die Plugin Development Tools helfen bei der Entwicklung von Plugins. |
| Restliche Bestandteile der Plattform | | Dazu gehören Plugins wie Text, Team, Debug uvm… |

* + 1. Vorteile

**TODO** Einfachere APIs, 20 things, Pojo 🡪 leichter zu testen, etc.

* + 1. Vergleich mit Eclipse RCP 3



Abbildung Architektur E3

Grundsätzlich sieht die Architektur von E3 ähnlich aus wie die von E4. Auch hier sind SWT und JFace im Einsatz. Equinox steht als OSGi Implementation zur Verfügung. Auch JDT, PDE und die restlichen Bestandteile der Plattform stehen zur Verfügung.

Der Unterschied bei E4 ist die Implementation der der Workbench (also das org.eclipse.org.workbench Plugin) und die neuen Technologien auf der diese neue Implementation basiert. Die Vorteile dieser neuen Technologien sind oben (Kapitel blabla) aufgeführt.

* + 1. Einschränkungen und Risiken

Verweis auf Risiken im Projektbericht?

* + 1. Qualität und Testbarkeit im Vergleich zu Eclipse RCP 3

Pojos vs. ViewParts etc..

**Wohin kommmt das?:**

**Option 1: Compatibility Layer einsetzen**

<http://eclipsesource.com/blogs/2012/06/18/migrating-from-eclipse-3-x-to-eclipse-4-e4/>

Der Compability Layer ermöglicht es Eclipse RCP 3.x Applikationen ohne Codeanpassungen auf der Eclipse 4 Plattform zu laufen. Wenn man nicht auf Eclipse 4 migriert so bleibt man kompatibel mit 3.x.

Um die Migration zu erleichtern bietet der Compability Layer the 3.x Workbench APIs an und übersetzt alle Aufrufe in das Programmiermodell von E4. Im Hintergrund wird transparent ein Application Model erstellt. Eclipse 3.x Applikationen sollten jedoch keine interne Workbench APIs benutzen um mit dem Compability Layer zu funktionieren.

Eclipse 4 benötigt aber zusätzlich die folgenden Plugins um zu funktionieren:

* org.eclipse.equinox.ds
* org.eclipse.equinox.event
* org.eclipse.equinox.util
* org.eclipse.e4.ui.workbench.addons.swt

Mit dem Compability Layer Ansatz alleine kann man aber nicht von den neuen Konzepten von E4 (Dependency Injection und Annotationen) Gebrauch machen. CSS funktioniert mit diesem Ansatz.

**Option 2: Eine reine Eclipse RCP 4 Applikation**

Diese Option wird nicht betrachtet, da ja eine bestehende Eclipse RCP 3.x Applikation migriert und nicht neu geschrieben werden soll.

**Option 3 Compatibility Layer und Eclipse RCP 4 Plugins**

Hier werden neue Plugins in Eclipse RCP 4 Manier geschrieben, die alten werden auf Eclipse 3.x belassen und laufen koexistent auf dem Compability Layer.

Es gibt 3 Arten Eclipse RCP 4 Plugins im Compability Layer zu integrieren.

1. Um dem Application Model – welches vom Compability Layer erstellt wird- Elemente hinzuzufügen werden Prozessoren und Fragmente benutzt. Hier gibt es aber aktuell noch Timing Probleme, denn wenn die Prozessoren und Fragmente verarbeitet werden hat der Compability Layer das Application Model noch nicht komplett erstellt. Diese Option mag für Handles und View funktionieren aber nicht für Editors.
2. Das Application Model, das vom Compability Layer erstellt wird, wird kopiert und als Application Model registriert. Diesem Application Model können nun neue Eclipse RCP 4 Komponenten hinzugefügt werden. Das Model XMI-File – konkret: LegacyIDE.xml - kann aus dem Plugin org.eclipse.ui.workbench herauskopiert werden.
3. **TODO:   
   Hier alle Schritte aufführen welche nötig sind um das Ganze (**LegacyIDE**.xmi) gebrauchen zu können.  
   Evtl. XMI splitten / Komponenten, wie???**
4. Die 3.x e4-Bridge von Tom Schindl ermöglicht es Views und Editoren in Eclipse RCP 3.x wie auch 4 zu benutzen. Um diese Bridge einzusetzen wird das Plugin org.eclipse.e4.tools.compat benötigt. Dieses Plugin stellt als Basisklasse einen Wrapper um Eclipse RCPS 4 POJOs zur Verfügung, diese Klasse DIViewPart erbt vom Eclipse 3 RCP ViewPart. Eine vollständige Anleitung ist unter <http://eclipsesource.com/blogs/2012/06/18/migrating-from-eclipse-3-x-to-eclipse-4-e4/> zu finden.

Fragment: Was ist zu tun (@see Teufel Pos. 3056)?

Abhängigkeit zu

org.eclipse.e4.ui.mode.workbench

org.eclipse.e4.core.di

javax.inject

Registrieren im plugin.xml

Z.B.

<extension

id="id1"

point="org.eclipse.e4.workbench.model">

<fragment

uri="fragment.e4xmi">

</fragment>

<processor

beforefragment="true"

class="ch.sbb.currency.e4.app.extension.ModelProcessor">

</processor>

</extension>

**Wohin mit dem?**

1. LegacyIDE.e4xmi
   1. Konkretes Beispiel RCS

Um eine Idee zu erhalten wie die oben aufgeführten Möglichkeiten zu bewerten sind habe ich mich entschlossen jede Option einmal auszuprobieren. Für die Migration habe ich einen möglichst einfachen ViewPart ausgewählt und diesen angepasst. Die Migrationsanleitungen und auch die Bewertung der jeweiligen Möglichkeit sind weiter unten aufgeführt.

Der einfache ViewPart sieht in der E3 Version folgendermassen aus:

public class ZugnummerRendererView extends ViewPart {

private GraphViewer graphViewer;

@Override

public void createPartControl(final Composite parent) {

final Composite composite = new Composite(parent, SWT.NONE);

composite.setLayout(new FillLayout());

graphViewer = new GraphViewer(composite);

graphViewer.setContentProvider(new ArrayContentProvider());

graphViewer.setInput(new Object[] { new Object() });

graphViewer.setViewPort(new WorldRectangle(0, 0, 100, 100));

graphViewer.setShapeProvider(new IShapeProviderImplementation());

}

@Override

public void setFocus() {

graphViewer.getControl().setFocus();

}

}

* + 1. Definition Abnahmekriterien

Damit dieser Aspekt als erfolgreich abgeschlossen gilt wurden vom Betreuer die folgenden Abnahmekriterien definiert:

1. Möglichkeiten aufgeführt und kritisch bewertet und beschrieben
2. Prototyp und Demo
   * 1. Migration
     2. Test
   1. Beschreibung des Aspektes „Adapters / Dependency Injection“

<http://wiki.eclipse.org/Eclipse4/RCP/EAS/List_of_All_Provided_Services>

Wenn field mit @Inject annotiert wird:

org.eclipse.e4.core.di.InjectionException: Unable to process "ZwlViewPart.site": no actual value was found for the argument "IWorkbenchPartSite".

Wenn in @PostConstruct angegeben:

Keine Fehlermeldung, aber Methode wird nicht aufgerufen

* 1. Diskussion der Eclipse RCP 4 Lösung
     1. Vorteile
     2. Vergleich mit Eclipse RCP 3
     3. Einschränkungen und Risiken
     4. Qualität und Testbarkeit im Vergleich zu Eclipse RCP 3
  2. Konkretes Beispiel RCS
     1. Definition Abnahmekriterien
     2. Migration
     3. Test
  3. Beschreibung des Aspektes „Commands / Handler, Menus, Key Bindings“
  4. Diskussion der Eclipse RCP 4 Lösung
     1. Vorteile
     2. Vergleich mit Eclipse RCP 3
     3. Einschränkungen und Risiken
     4. Qualität und Testbarkeit im Vergleich zu Eclipse RCP 3
  5. Konkretes Beispiel RCS
     1. Definition Abnahmekriterien
     2. Migration
     3. Test
  6. Beschreibung des Aspektes „Eigene Extension Points / Eigene Services“
  7. Diskussion der Eclipse RCP 4 Lösung
     1. Vorteile
     2. Vergleich mit Eclipse RCP 3
     3. Einschränkungen und Risiken
     4. Qualität und Testbarkeit im Vergleich zu Eclipse RCP 3
  8. Konkretes Beispiel RCS
     1. Definition Abnahmekriterien
     2. Migration
     3. Test
  9. Beschreibung des Aspektes „Application Model vs. Advisors“
  10. Diskussion der Eclipse RCP 4 Lösung
      1. Vorteile
      2. Vergleich mit Eclipse RCP 3
      3. Einschränkungen und Risiken
      4. Qualität und Testbarkeit im Vergleich zu Eclipse RCP 3
  11. Konkretes Beispiel RCS
      1. Definition Abnahmekriterien
      2. Migration
      3. Test

1. Reflexion