KSM: Eclipse RCP

Studienarbeit 2

Studiengang Informationstechnik
an der Dualen Hochschule Baden-Württemberg
Standort Stuttgart Campus Horb
von

Yves Fischer

Abgabedatum:
20. Mai 2011

Kurs	TIT2008/NS
Betreuer der Dualen Hochschule	Dr. Jörg Wedeck

Erklärung

Ich habe die Studienarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Rheinau, 20. Mai 2011

_____

Zusammenfassung

Im vorangegangen Semester wurde die Portierung der KSM Java/Swing Applikation auf Eclipse RCP evaluiert.

In dieser Arbeit sollen konkrete Vorgehensweisen erörtert und der Weg zu einer Eclipse RCP basierten KSM Applikation geebnet werden.

Motivation

Das GUI von KSM basiert momentan auf Swing bei Verwendung des NetBeans GUI--Designers. Der auf diesem Weg erzeugte Code ist nahezu unwartbar und auch durch die ständige Weiterentwicklung inkonsistent.

Mithilfe von Eclipse RCP können dank klarer Strukturen und Konventionen Verbesserungen im Bereich Wartbarkeit, Erweiterbarkeit und Usability erzielt werden.

Problemstellung und Ziele

Während die vorhergehende Studienenarbeit [1] sich mit der Einarbeitung in Eclipse RCP und der grundsätzlichen Möglichkeit der Realisierung von KSM darin beschäftigte, soll mit dieser Studienarbeit die konkrete Umsetzung begonnen werden.

Summary

The previous assignment evaluated the porting of the KSM Java / Swing Application to Eclipse-RCP.

This follow-up discuss specific ways to an Eclipse RCP based KSM application.

Motivation

Currently KSM is based on a Java-Swing Graphical User Interface, developed with the NetBeans GUI-Designer. This GUI-related code is inconsistent and unmaintainable due to continuos development.

Its possible to gain a better maintainability, expandability and usability by using Eclipse RCP through clear structures and conventions.

Tasks and Objectives

While the foregoing Studienenarbeit elaborates basic Eclipse-RCP technics and presented a rough-prototype, its now time to begin with the implementation of specific points of a new KSM Application in Eclipse-RCP to supersede the Swing-based Application.

The ongoing development will be in close cooperation with students working on bugfixing and extending the Swing-based KSM, to gain interoperability and quality.

Inhaltsverzeichnis

1 Einleitung

Aufbauend auf den Ergebnissen der vorangegangen Studienarbeit gilt es zu beginnen, mithilfe von Eclipse RCP ein KSM Werkzeug zu erstellen, welches auf längere Sicht den die Java/Swing Version ablösen kann.

Für ein KSM auf Basis von Eclipse RCP sprechen viele Gründe von denen einige im folgenden kurz erläutert werden:

Erweiterbarkeit/Modularität Die Entwicklung von KSM hat viele verschiedene Versionsstände hervorgebracht die oftmals nicht wieder vereint werden konnten. Wäre das Grundprojekt modularer aufgebaut gewesen hätten diese einfacher in ein großes Ganzes integriert werden können.

Zu diesem Zeitpunkt existieren zwei im Grunde gleiche aber getrennte KSM Entwicklungszweige: KSM und QKSM. Der Ideen und Bugfix Austausch zwischen den Entwicklern und Projektständen ist durch die strikte Trennung nahezu null.

Wenn zukünftig weitere Anwendungen mit ähnlichen Anwendungsbereich in Eclipse RCP entwickelt werden ist es möglich diese mit wenig Aufwand gebündelt auszuliefern.

Vereinfachung Durch Verwendung von Softwarekomponten und Frameworks aus der Eclipse Plattform kann auf aufwendige, fehleranfällige Eigenentwicklungen verzichtet werden.

Bessere GUI Für Benutzer von Eclipse wird das Arbeiten mit Workspaces und Projekten in der Eclipse Art bereits vertraut sein. Alle anderen werden mit einem intuitiven Standard- — weniger individuell-kreativen — Oberfläche konfrontiert.

Lizenzmanagement KSM soll in naher Zukunft öffentlich verfügbar sein, QKSM nicht.

Wäre QKSM ein Zusatzmodul (Bundle, Plugin) für KSM, dann würde lediglich eine zusätzliche Datei einem Open-Source KSM QKSM Fähigkeiten verleihen.

Deployment/Update Eclipse verfügt über ausgereifte Mechanismen zur Softwareverteilung und Aktualisierung. Über eine Update-Site könnten Benutzer über das Internet Ihre KSM-Installation modular Ihren Bedürfnissen anpassen und aktualisieren.

Web Anwendungen Mithilfe von RAP (Rich Ajax Platform) lassen sich RCP Anwendungen ins Web portieren.

Nicht zuletzt soll die wertvolle Erfahrung des Umgangs mit einer „open source, robust, full-featured, commercial-quality, industry platform for the development of highly integrated tools and rich client application" erwähnt werden.

Diese Studienarbeit wird die Vorteile der Nutzung der Eclipse Plattform nur zum Bruchteil auskosten. Nicht zu vergessen ist, dass es auch Hindernisse gibt auf die im Kapitel 4 ab Seite 21 eingegangen wird.

Zu Beginn wird die Entwicklung eines Datenformat für die Computerdarstellung von Kybernetischen System-Modellen besprochen. Kapitel 3 beschreibt Veränderungen gegenüber dem Prototyp aus der vorhergehenden Arbeit.

In Kapitel 4 wird auf das Erlernen des Umgangs und die Arbeit mit Eclipse RCP/PDE besprochen. Kapitel 5 wird die Ergebnisse dieser Arbeit kurz zusammenfassen und zur möglichen Weiterentwicklung Stellung nehmen.

2 Datenformat und Datenmodell

In der Studienarbeit 1 wurde mit einem vereinfachten Datenmodell gearbeitet.

Da zukünftig sowohl KSM/Swing als auch /RCP die gleichen Daten verarbeiten können sollten, liegt es nahe, dass das Format in dem die Daten persistent im Dateisystem abgelegt sind vereinheitlicht werden sollte.

Im folgenden wird ausgeführt warum das aktuelle Datenformat in KSM/Swing nicht geeignet ist und wie ein neues entwickelt wurde.

> In Anhang 5.2 ist die vollständige Dokumentation des neuen Datenformat zu finden dessen Entwicklung in diesem Kapitel beschrieben wird.

2.1 Istzustand

In der Evaluation von RCP wurde ein einfaches Datenformat eingeführt, indem die Model-Objekte einfach mittels leicht annotierten JAXB serialisiert wurden [1, S. 22f]:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<diagram>
    <connections>
        <connection>
            <source>1c862df6-80e5-445b-a41c-ed677973abfb</source>
            <target>e36a0368-d8f6-40a2-bd29-375352045da8</target>
        </connection>
    </connections>
    <nodes name="Node 01">
```

```
        <id>1c862df6-80e5-445b-a41c-ed677973abfb</id>
        <location>
             <x>128</x>
             <y>90</y>
        </location>
        <nodeProperties/>
    </nodes>
    <nodes name="Node 11">
        <id>e36a0368-d8f6-40a2-bd29-375352045da8</id>
        <location>
              <x>531</x>
             <y>104</y>
        </location>
        <nodeProperties/>
    </nodes>
</diagram>
```

In der aktuellen KSM/Swing Applikation wird ebenfalls ein auf XML-basierendes Datenformat verwendet.

Abbildung 2.1 zeigt ein einfaches KSM-Diagramm in KSM/Swing. Listing 2.1 gibt sehr stark gekürzt das bei der Speicherung erzeugte XML wieder.

Listing 2.1: Datenformat aus KSM/Swing Applikation Stand 22.02.2011

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<KyberneticSystemModel>
        <nodes>
                <node>
                        <node_ID>0</node_ID>
                        <x_pos>469</x_pos>
                        <y_pos>423</y_pos>
                        <name>new_Node_0</name>
                        <comment />
                        <allowModeration>false</allowModeration>
                        <moderationsteps>1</moderationsteps>
                        <Moderation>1.0</Moderation>
                        <adaptionRate>0.2</adaptionRate>
                        <Reaction>1.0</Reaction>
                        <moderationMethod>linear</moderationMethod>
                        <color>White</color>
                        <std_height>50</std_height>
```

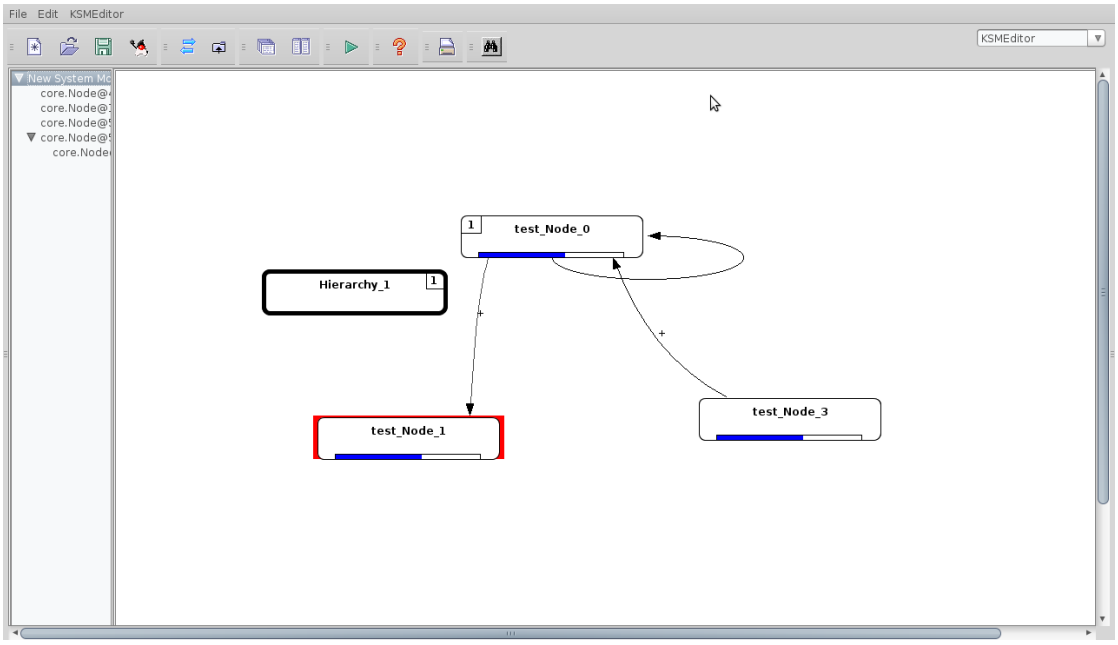

Abbildung 2.1: KSM/Swing Applikation bei der Anzeige der Daten aus Listing 2.1

```
<std_width>220</std_width>
<bShowProgressBar>true</bShowProgressBar>
<bCustomizeNodeTransparent>false</bCustomizeNodeTransparent>
<bCustomizeNodeBorder>true</bCustomizeNodeBorder>
<iTextVerticalFactor>33</iTextVerticalFactor>
<sIconPath />
<scaleFct>
</scaleFct>
<belongsToHierarchy>0</belongsToHierarchy>
<hierarchyDepth>0</hierarchyDepth>
<useScaleFct>false</useScaleFct>
<inWeightEdges>
        <inWeightEdge>
                <tupel_value>0.0</tupel_value>
        </inWeightEdge>
        <inWeightEdge>
                <tupel_value>1.0</tupel_value>
        </inWeightEdge>
        <inWeightEdge>
                <tupel_value>0.0</tupel_value>
        </inWeightEdge>
</inWeightEdges>
<UserValue>
        <tupel_value>3.0</tupel_value>
</UserValue>
```

```
                <MinValue>
                        <tupel_value>0.0</tupel_value>
                </MinValue>
                <MaxValue>
                        <tupel_value>5.0</tupel_value>
                </MaxValue>
                <extern>
                        <tupel_value>0.0</tupel_value>
                </extern>
                <scaleCoeffizient>1.0</scaleCoeffizient>
        </node>
        <node>
                <node_ID>1</node_ID>
                <x_pos>299</x_pos>
                <y_pos>216</y_pos>
                <name>new_Node_1</name>
                ...
        </node>
</nodes>
<arrows>
        <arrow>
                <start_node>1</start_node>
                <end_node>0</end_node>
                <start_pos>6</start_pos>
                <end_pos>1</end_pos>
                <bend_xpos>497</bend_xpos>
                <bend_ypos>313</bend_ypos>
                <saved>true</saved>
                <funcType>gui.functions.FuncStraightLine</funcType>
                <arrowfunction>
                        <m>1.0</m>
                        <b>0.0</b>
                        <position>0</position>
                        <FuncDraggingPoints>
                                <FuncDraggingPoint>
                                        <moveHorz>false</moveHorz>
                                        <moveVert>true</moveVert>
                                        <first>true</first>
                                        <last>false</last>
                                        <funcPx>0.0</funcPx>
                                        <funcPy>0.0</funcPy>
                                </FuncDraggingPoint>
                                <FuncDraggingPoint>
                                        <moveHorz>false</moveHorz>
                                        <moveVert>true</moveVert>
                                        <first>false</first>
                                        <last>true</last>
                                        <funcPx>1.0</funcPx>
                                        <funcPy>1.0</funcPy>
                                </FuncDraggingPoint>
```

```
                        </FuncDraggingPoints>
                    </arrowfunction>
                </arrow>
                <arrow>
                    <start_node>0</start_node>
                    <end_node>2</end_node>
                    <start_pos>3</start_pos>
                    <end_pos>8</end_pos>
                    <bend_xpos>674</bend_xpos>
                    <bend_ypos>388</bend_ypos>
                    <saved>true</saved>
                    <funcType>gui.functions.FuncStraightLine</funcType>
                    ...
                </arrow>
        </arrows>
        <hierarchies />
</KyberneticSystemModel>
```

Für das KSM/Swing Datenform existiert ein XML-Schema in der Ausarbeitung der Studienarbeit von Friedhelm Wolf [2] in Anhang 4. Es wurde aber in den folgenden Studienarbeiten nicht mehr weiter gepflegt und ist daher vermutlich nicht mehr gültig.

Darüberhinaus existierte keine saubere Dokumentation dieses Datenformates. Die Bedeutung der abgelegten Werte muss aus dem Quelltext von KSM/Swing interpretiert werden - was nicht immer einfach ist.

Weiterhin scheinen die Konventionen beim Entwurf dieses Formates (oder „Schema") eher beliebig gewesen zu sein. Dies fällt als erstes auf bei der Benennung der der Eigenschaftsnamen (Tags), wo `CamelCase` („Microsoft Stil"), `javaStilArt` und underline_stil wild gemischt werden. Die Benamung verwendet teils einen Typ-Prefix (`sIconPath`), teils auch nicht.

Die Ein- und Ausgabe erfolgt in KSM/Swing mithilfe der Java-Bibliothek *com.machinedialog*. Von Seiten der Projektbetreuung wurde geäußert, dass es gewünscht ist diese alte Bibliothek nicht mehr weiter zu verwenden. Zu den genauen Gründen sei auf die Studienarbeit 1 von Tobias Dreher verwiesen [3, S. 24].

Die fehlende Dokumentation und der anscheinend unstrukturierte Entwurf sprachen ebenfalls gegen die Weiterverwendung dieses Datenformates. In Sicht zur Zukunft ist das Format ungeignet, weil es nicht erlaubt, dass zusätzliche Eigenschaften rückwärtskompatibel abzubilden.

Weiterhin ist es ist nicht (mehr) durch ein XML-Schema gestützt und es wurde im Laufe der Entwicklung mehrmals beliebig verändert. Es ist nicht dokumentiert und der Parser in KSM/Swing Programm ist in einem schlechten, mit der GUI verknüpften Zustand.

2.2 Entwicklung eines neuen Datenformat

Wie an diesen beiden Beispielen zu sehen ist, bietet das XML-Format aus der Studienarbeit 1 nicht das Ausdrucksvermögen des KSM/Swing Format. Letzteres ist jedoch ungeeignet zur Nutzung in einer neuen KSM RCP-Applikation.

Daher lag es nahe ein Format zu entwerfen welche sowohl von KSM/Swing mithilfe einer sauberen neuen Import/Export Infrastruktur als auch in der RCP-Anwendung genutzt werden kann. Es wurde hierbei ein Ansatz gewählt der sowohl Knoten, Gruppen von Knoten (auch bekannt als „Hirarchien") als auch gerichtete Verbindungen zu einem anderen Knoten abbilden kann. Alle drei Objekte können durch Eigenschaften dynamisch, nicht schema-gebunden mit Informationen angereichert werden.

> Die Implementierung des neuen Datenformat in **KSM/Swing** wurde von Tobias Dreher vorgenommen. Siehe dazu seine Studienarbeit [4].

Zu festen Standardisierung wurde ein XML-Schema erstellt woraus wiederum mittels des XML-Schema-Compiler (xjc) von JAXB Java-Klassen erzeugt werden die mit weiteren, handgeschrieben Klassen eine Bibliothek zur Abbildung von KSM-Diagrammen im Speicher sowie Funktionen zum Laden und Speichern bietet (Abb. 2.2). Diese kann sowohl von KSM/RCP als auch beliebigen Java-Anwendungen verwendet werden.

Die Datenmodell Bibliothek (Name `de.dhbw.horb.ksm.xmlschema`) ist als Eclipse-Project angelegt kann jedoch auch nur mit Apache-Ant verwaltet werden (ist somit Netbeans kompatibel). Das Projektverzeichnis enthält weiterhin umfassende Unit-Tests, teils im Beaviour-Driven Stil mithilfe der Bibliothek *mockito*.

Listing 2.2 zeigt valide Beispieldaten zu dem im Anhang ab Seite 38 hinterlegten, neuen XML-Schema.

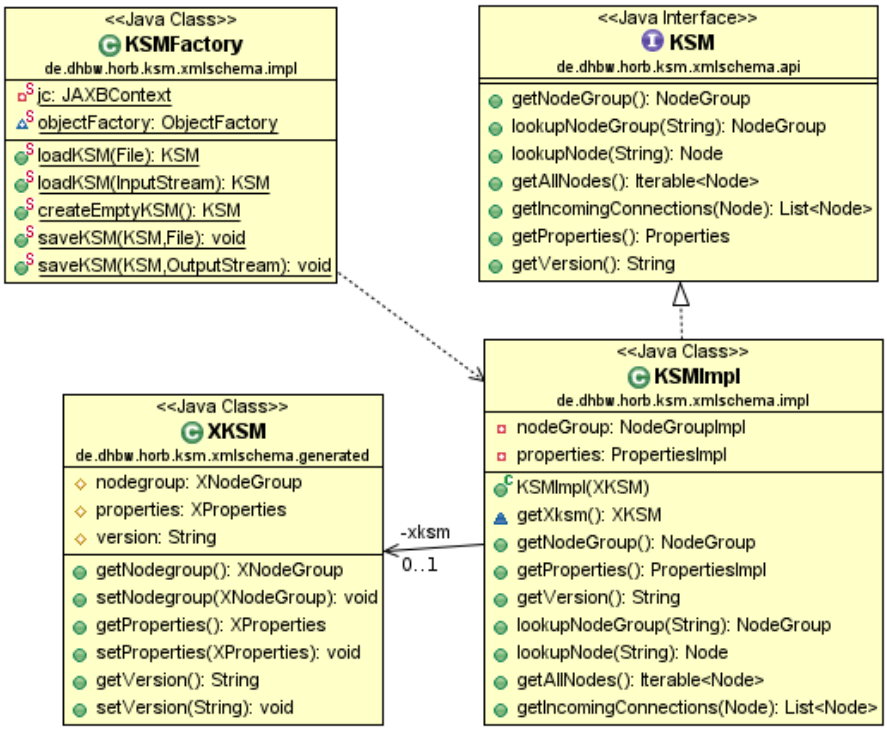

Abbildung 2.2: Klassendiagramm XML-Datenmodell Bibliothek, Ausschnitt KSM Model-Klasse

Listing 2.2: KSM Daten im neuen Datenformat (example-1.xml)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ksm xmlns="http://www.ba-horb.de/~ksm/xml/ksm-1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ba-horb.de/~ksm/xml/ksm-1 ksm-1.xsd"
  version="1.0">
  <nodegroup id="root">
    <node id="56-7567-567567-657657-567">
      <properties>
        <string name="visual.caption">a node</string>
        <string name="visual.color">#ffeecc</string>
        <decimal name="visual.location.x">420</decimal>
        <decimal name="visual.location.y">420</decimal>
        <stringList name="meineTolleList">
          <value>TestValue</value>
        </stringList>
        <decimalList name="nochEine">
          <value>1.2</value>
        </decimalList>
      </properties>
      <connections />
    </node>
```

```
    <nodegroup id="0123123-123123-123123-123213">
      <node id="56-7567-12345-657657-567">
        <properties>
          <string name="visual.color">#ffeecc</string>
          <decimal name="visual.location.x">420</decimal>
          <decimal name="visual.location.y">420</decimal>
        </properties>
        <connections>
          <connection to="56-7567-567567-657657-567">
            <properties>
              <string name="visual.caption">A Connection</string>
              <string name="visual.color">#ffeecc</string>
              <decimal name="data.weight">5</decimal>
            </properties>
          </connection>
        </connections>
      </node>
      <properties>
        <string name="visual.caption">a Hierarchy</string>
        <string name="visual.color">#ffeecc</string>
        <decimal name="visual.location.x">420</decimal>
        <decimal name="visual.location.y">420</decimal>
      </properties>
    </nodegroup>
    <properties />
  </nodegroup>
</ksm>
```

Weitere Dokumentation zum Format und der Bibliothek finden sich im Anhang ab Seite 28.

3 Weiterentwicklung Programmfunktionalität und Modularisierung

Das KSM/RCP Projekt besteht aus folgenden Komponenten:

`de.dhbw.horb.ksm.core` Plugin Project, Kernfunktionalität:

- RCP-Anwendung; Einstiegspunkt
- Editor auf Basis von GEF (Figures, EditParts, Policies, Commands)
- Projekttyp (*Nature*), Projektwizard, Outline und Navigator
- KSM-Properties-Description Extension-Point

`de.dhbw.horb.ksm.qksm` Fragment Project von `.core`, enthält nur eine Demo für den Property Extension-Point.

`de.dhbw.horb.ksm.simulator` Plugin Project, Enthält eine Demo der Live-Verknüpfung des im Editor dargestellten Modells mit einem Chart auf Basis von SWTChart.

`de.dhbw.horb.ksm.tableeditor` Plugin Project, enthält einen Prototyp zum Editieren von Knoteneigenschaften in Tabellenform.

`ksm-model` Java-Project/Apache Ant, Datenmodell:

- XML-Schema des Datenformat.
- Java Library auf Basis von JAXB.

- Dokumentation des Datenformat.

de.dhbw.horb.ksm.model Plugin Project welches ksm-model als OSGi-Bundle zur Verfügung stellt. Muss nach Aktualisierung von ksm-model ebenfalls aktualisiert werden.

Das ...qksm-Projekt ist als Eclipse-Projekt vom Typ „Plugin-Fragment" abgelegt. Das bedeutet, es hängt direkt vom KSM-Core Plugin als Host-Plugin ab und ist kein eigenständiges (OSGi-)Bundle. Es wird wird beim Start mit dem Host-Plugin vereinigt. Diese vorgehensweise eignet sich um ein Plugin ohne es zu verändern nachträglich um Features wie z.B. Internationalisierung oder plattformabhängigen Code zu erweitern.

Ein Fragment Projekt unterscheidet sich von einem Plugin Projekt in weiteren Punkten. Da es kein eigenständiges Bundle ist, besitzt keine Activator-Klasse. Die Datei zur Beschreibung der Extensions heißt fragment.xml anstatt plugin.xml. In META-INF/MANIFEST.MF wird daher kein Activator deklariert - jedoch mit Fragment-Host das zu erweiternde Plugin.

Das ...model Projekt wurde durch den Eclipse Assistenten „Plug-In from existing JAR" erstellt. Es enthält die .class Dateien aus dem JAR das im Projekt ksm-model durch ant erzeugt wird. Die zukünftige aktualisierung kann einfach dadurch erfolgen, dass die Class-Files von Hand in das ...model Projekt kopiert werden. Dadurch dass man auf diesen Weg das ksm-model Projekt als OSGi Bundle zur Verfügung hat können KSM/RCP-erweiternde Plugin dieses einbinden. Alternativ wäre es möglich gewesen die .jar-Datei aus ksm-model in das ...core-Projekt einzubinden und von dort aus zu exportieren. Damit müsste jedoch mit jeder Änderung in ksm-model das ...core-Plugin „angefasst" werden.

Das ksm-model Projekt wurde aus Rücksicht auf die nicht-Eclipse Entwicklung frei von Eclipse-Abhängigkeiten gehalten und nicht gleich als OSGi-Bundle implementiert.

3.1 Die RCP Applikation `ksm.core`

Eine grafische Eclipse RCP Anwendung verfügt immer über ein zentrales Plugin als Einstiegspunkt in die Ausführung. Diese implementiert das Interface `IApplication` und registriert sich auf den Extension-Point `org.eclipse.core.runtime.applications`. Die `IApplication`-Klasse erzeugt den `WorkbenchAdvisor` welcher das grundlegende Aussehen mithilfe von Perspektiven und anhand einem `WorkbenchWindowAdvisor` steuert [5].

In KSM/RCP ist im Kern-RCP Plugin die gesamte KSM-Editierfunktionalität enthalten. Dazu gehört der GEF-basierte Editor, das Outline und die Projektansicht.

3.1.1 Der grafische Editor und das Datenmodell

Der mit dem Graphical Editing Framework (GEF) umgesetzte Editor ist teil des `...ksm.core` Plugin-Projekt. GEF geht von einem existierenden Datenmodell aus welches hier durch das Plugin-Project `...ksm.model` anstelle von `ksm-model` zur Verfügung gestellt wird [6].

Bei der „Initialisierung" eines GEF-Editors wird dem Editor eine Factory zugewiesen die für die Objekte/verschiedenen Typen des Datenmodells GEF-EditParts erzeugen kann (`DiagramEditor#configureGraphicalViewer` (Abb. 3.1).

Anschliessend wird des Datenmodell zugewiesen (`DiagramEditor#initializeGraphicalViewer`). Der GEF-Editor ruft nun mit dem „Parent"-Objekt des Datenmodells die `PartFactory` auf und erhält das entsprechende EditPart. Der weitere Abstieg in der Hirarchie des Datenmodells geschieht durch den Aufruf von `GraphicalEditPart#getModelChildren()` welcher eine Liste von Kind-Elementen des jeweiligen Elementes ergibt. So liefert bspw. die KSMEditPart hier alle NodeGroup- und Node-Elemente aus der Root-NodeGroup des Datenmodels.

Das GEF Programmiermodel ist dem *Model-View-Controller* Pattern angelehnt. Abbildung 3.2 zeigt am Beispiel des KSM-Typ aus dem Datenmodell die Zugehörigkeit der Klassen zu Model, View und Controller.

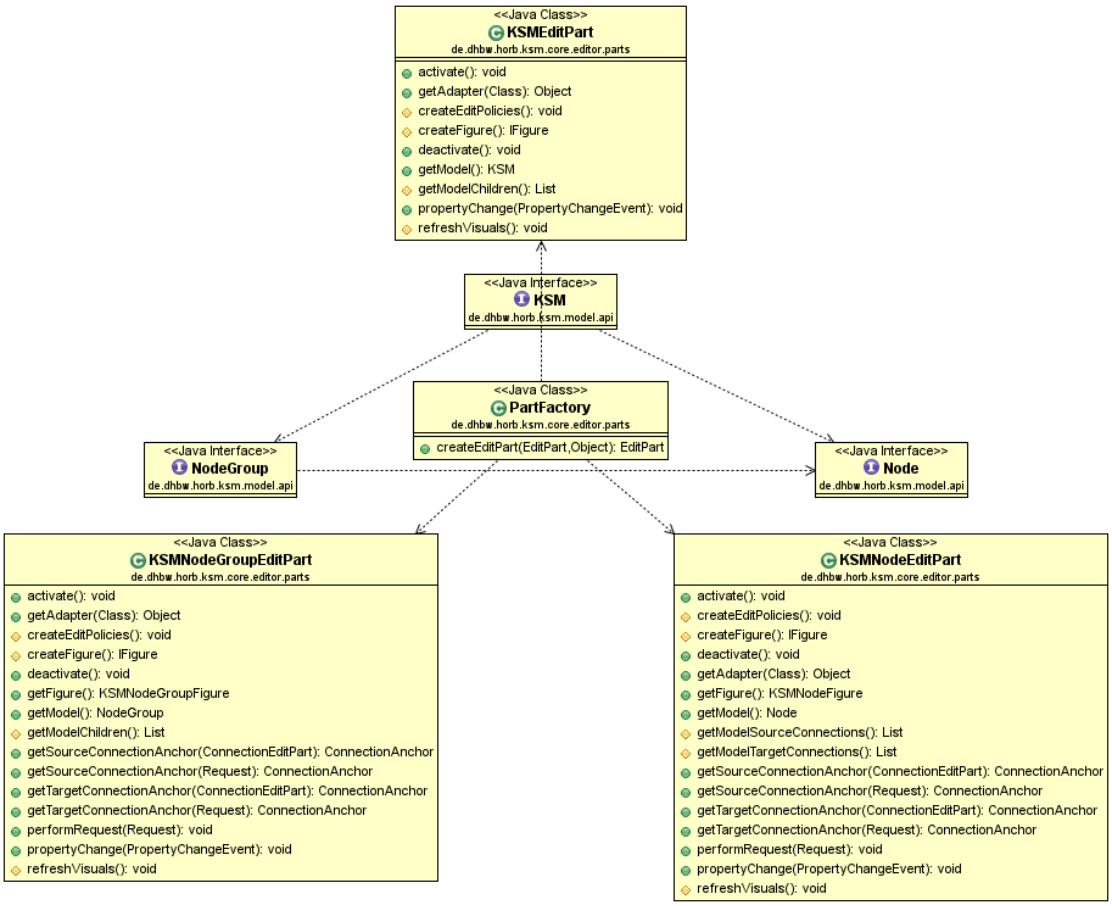

Abbildung 3.1: Model und View von KSM/NodeGroup/Node

Die Edit-Policies erfüllen die Rolle des Controllers indem sie definieren, was passieren soll wenn bestimmte Umstände eintreten. U.a. falls der Benutzer angefordert hat das Element zu löschen (z.B. durch Drücken der Entfernen Taste) oder einen „Direct-Edit" durch einen langsamen Doppel-Klick angefordert hat.

3.1.2 Commands

Alle Manipulationen des Datenmodells werden durch sogenannte `Commands` beschrieben. Abbildung 3.3 zeigt die Basisklasse und abgeleitete Klassen die Manipulationen des `Node`-Typ beschreiben.

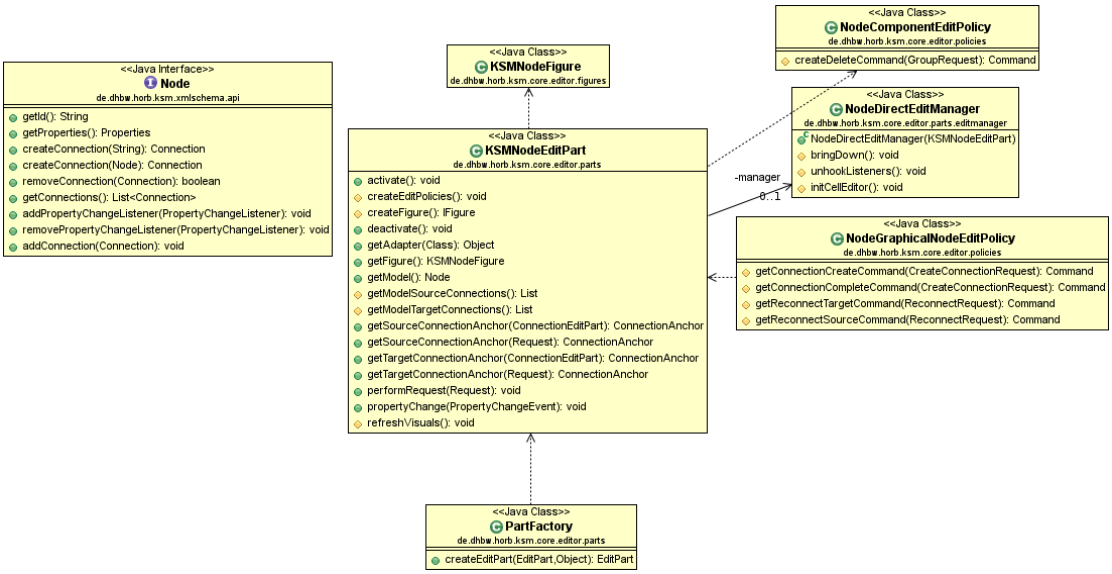

Abbildung 3.2: Model, View und Controller des Node-Typ (von Rechts nach Links)

Commands implentieren jeweils die Durchführung der Manipulation als auch das Rückgängig machen. Durch GEF werden die Commands bei Anwendung auf einem Stack abgelegt wodurch das klassische Undo/Redo ermöglicht wird.

Dieses Pattern verhindert weiterhin, dass das Datenmodell an beliebigen Code-Stellen manipuliert wird trägt so zur allgemeinen Wartbarkeit bei.

3.2 Simulation

Das „Simulation" Project `...ksm.simulator` zeigt als Prototyp wie es möglich ist, Daten die im Editor bearbeitet werden live mit einem Chart auszuwerten.

Der Chart ist als View implementiert. Diese View besitzt für mehrere Zustände für alle KSM-Editoren die geöffnet sind und wird entsprechend aktualisiert.

Für den Linechart wird die Bibliothek SWT-Chart verwendet die dem Projekt beigelegt ist. Sie wird im OSGi-Manifest angegeben:

...

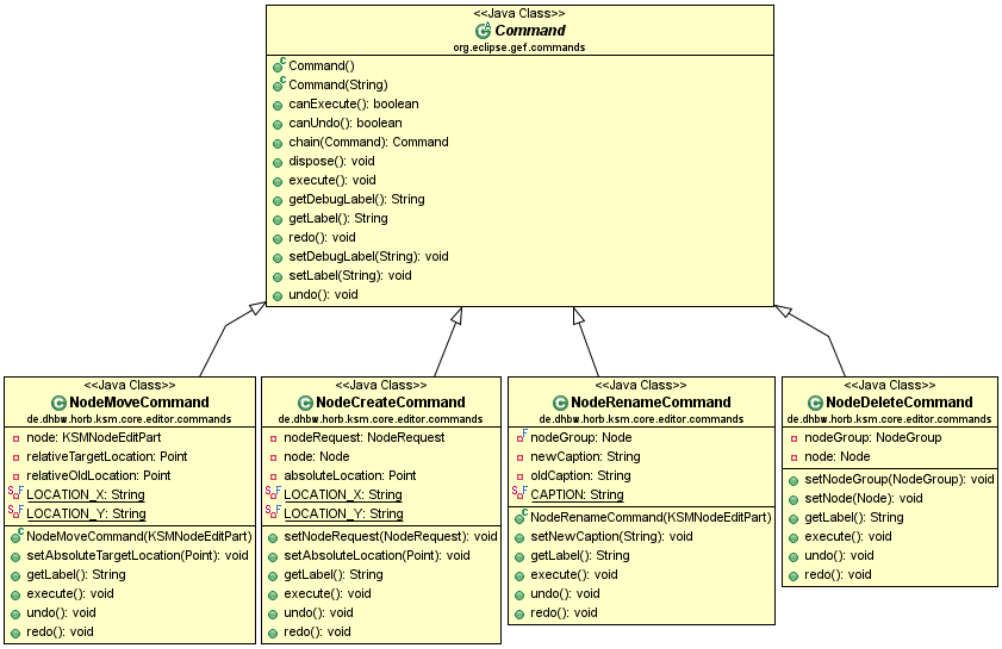

Abbildung 3.3: Command Basisklasse und Commands betrefft dem `Node`-Typ

```
Bundle-ClassPath: lib/org.swtchart_0.7.0.v20110128.jar,
 lib/org.swtchart.ext_0.7.0.v20110128.jar,
 .
```

Der Prototyp der Simulationsdarstellung trägt lediglich die X- und Y-Position des KSM-Nodes auf die Y-Achse auf und die KSM-Node-Beschriftung auf die X-Achse.

3.3 Property Editor

Damit ein Model für die Standard Eclipse-Properties-View nötigen Informationen bereitstellt muss eine `PropertySource` bereitgestellt werden. Dies geschieht im einfachen Fall indem die Model-Klasse `IPropertySourceProvider` implementiert.

Ist dies nicht der Fall fragt die Properties-View das `EditPart` über `getAdapter(IPropertySourceProvider.class)` nach einem Objekt das die Properties des Model beschreibt. Dieser Ansatz wurde gewählt, da die Klassen des Models nicht angepasst

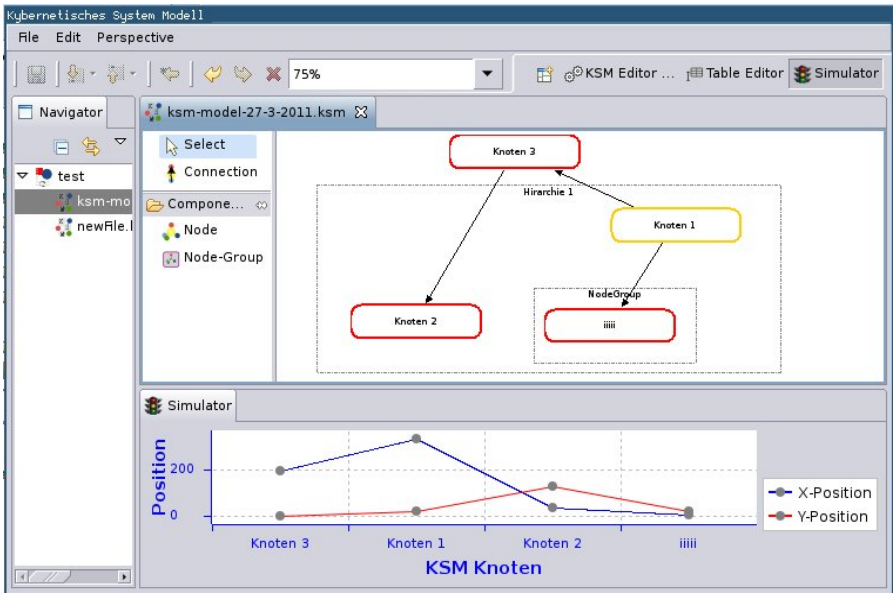

Abbildung 3.4: Simulator Prototyp

werden konnten. Dies ist in einer Designschwäche der Model-Bibliothek begründet. Es ist noch nicht möglich eigene, abgeleitete Klassen für das Model zu verwenden.

Es wurden daher die Klassen `...core.editor.model.property.*` als PropertySourceProvider für `Node` und `NodeGroup` eingeführt. Abbildung 3.5 zeigt als UML-Klassendiagramm die Klassen zur Verwaltung der Properties.

Für jedes Model-Objekt meldet das entsprechende EditPart eine Instanz von `ModelPropertySource`. Das Attribut `type` ist dabei je nach Model eines von `ksm, nodegroup, node, connection`. Anhand dieses Typs wird in der ExtensionRegistry nachgeschlagen ob für den ExtensionPoint `de.dhbw.horb.ksm.core.model.property` Extensions registriert wurden.

Extensions sehen so aus, dass eine von `AbstractPropertyDescriptorAdvisor` abgeleitete Klasse (im Beispiel sind schon BaseNode[Group]PropertyAdvisor dargestellt die die Manipulation der grundlegenden Eigenschaften Farbe und Beschriftung erlauben) angegeben wird die Descriptoren für Eigenschaften des Model-Objekt erstellt.

Abbildung 3.6 zeigt wie der Extension Point deklariert wurde. Das gezeigte GUI ist eine Maske für eine XML-Datei die im Prinzip ein XML-Schema für das XML ist mit dem

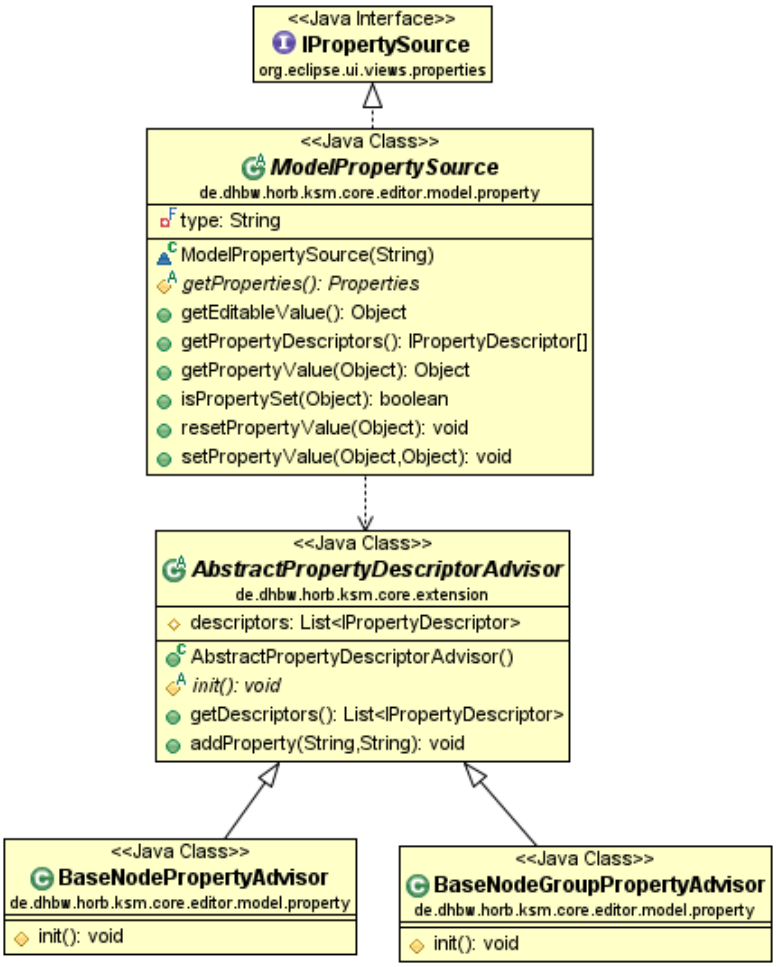

Abbildung 3.5: Klassen des Extension Point für Property Deskriptoren

der Extension-Point genutzt werden kann. In diesem Fall können auf den Extension-Point beliebig viele „advisor" Elemente angelegt werden die auf eine von `AbstractPropertyDescriptorAdvisor` abgeleitete Klasse zeigen und ein Attribut `type` haben (Abb. 3.7).

3.4 Table-Editor

Der Table-Editor „besteht aus sechs Reitermenüs. Diese besitzen folgende Funktionalitäten: [Editoren für] ‚Edge Values' – Kanteneigenschaften [..] ‚Node Values' – Knoteneigenschaf-

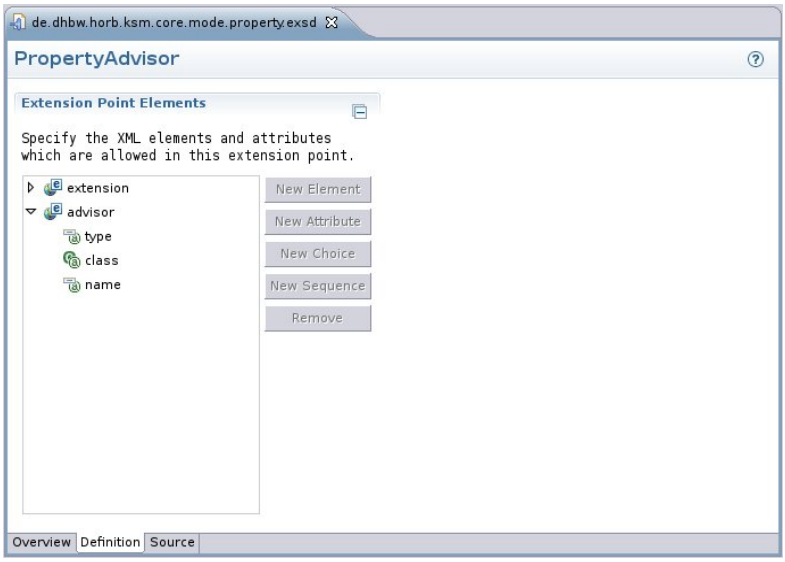

Abbildung 3.6: Deklaration des Extension-Point für Property-Advisors

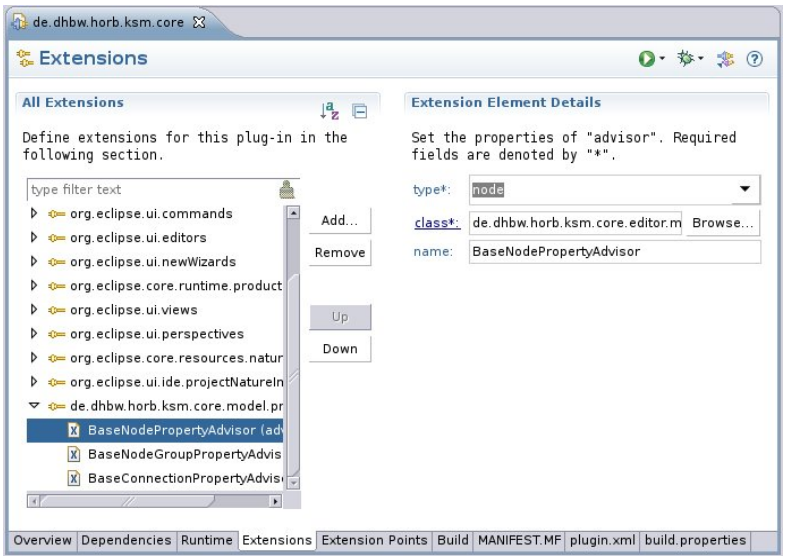

Abbildung 3.7: Benutzung des Extension-Point für Property-Advisors

ten" (Studienarbeit Christian Riess [7, S. 24]).

Im Rahmen dieser Studienarbeit wurde ein Table-Editor prototypisch entworfen welcher Knoteneigenschaften manipulieren kann. Abbildung 3.8 zeigt die dabei eingeführten Neuigkeiten.

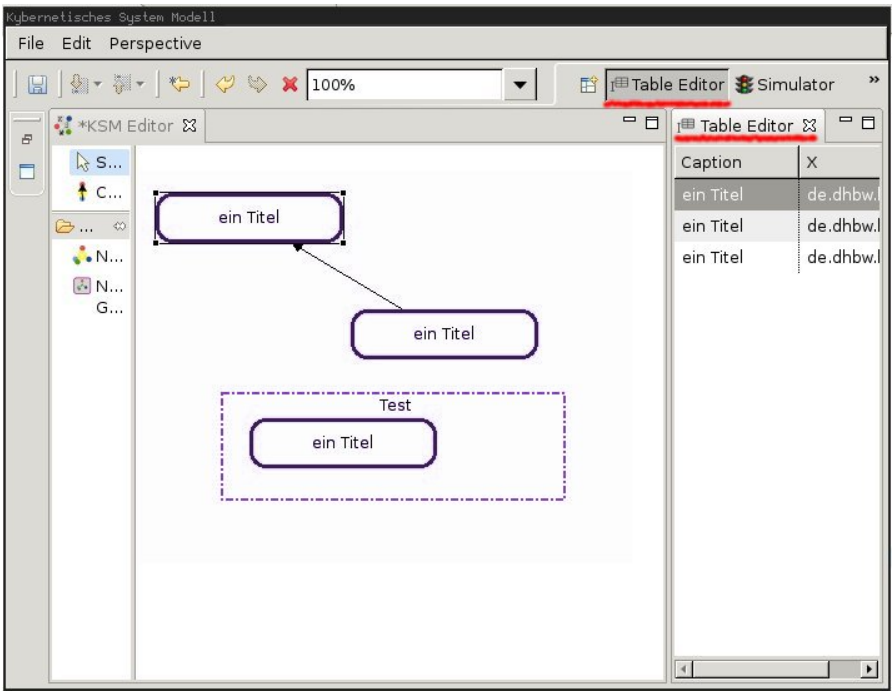

Abbildung 3.8: Prototyp eines Table-Editor

Rot markiert ist die neue View des „Table-Editor". Sie wird durch den Aufruf der Table-Editor Perspektive aktiviert (Button ebenfalls rot markiert).

Bei der View handelt es sich `PageBookView`. Sie instanziert für jeden offenen Editor einen Table-Editor. Es wird jeweils der zu dem fokussierten Diagramm-Editor passende Table-Editor angezeigt. Die Implentierung folgt dabei dem Eclipse-Standard wie er auch in Outline verwendet wird.

Änderungen am Diagramm übernimmt der Table-Editor einfacherweise indem er auf alle Node/NodeGroup Objekte listener registriert sondern er beobachtet den Command-Stack des Editors. Dies funktioniert nur solange wie keine weiteren Änderungen am Modell vorgenommen werden die kein Command im Diagramm-Editor Command-Stack erzeugen.

Der Table-Editor nutzt im Prototyp noch keine Undo-/Redo-Funktionalität sondern arbeitet „dreckig" direkt am Datenmodell.

4 Eclipse RCP Programmierung

Dieses Kapitel soll zukünftigen Entwickler einige Hinweise zum schnellen Start geben. Dazu soll die in dieser Arbeit verwendete Arbeitsumgebung vorgestellt und die dabei gewonnenen Erfahrungen bei der Einarbeitung und Nutzung mitgeteilt werden.

Weiterhin werden Vorschläge zur stärken Integration von Eclipse-Tools in das Studium aus der Sicht eines Studenten vorgestellt.

Es bei dieser Studienarbeit mit dem Eclipse 3.6 / Helios Package „for RCP and RAP Developers" gearbeitet.

4.1 Hilfsmittel

Zu Beginn der Studienarbeit I verfügte ich über Basiskentnisse der Eclipse Java Development Tools (JDT). Das Thema Eclipse RCP wurde im Studium nicht angesprochen und daher war meine primäre Aufgabe einen Einstieg in das Thema RCP-Entwicklung zu finden.

Zum Kennenlernen von Eclipse ist der Besuch einer Eclipse Demo Camp Veranstaltung empfehlenswert. Dort werden neue, auf Eclipse aufbauende Technologien vorgestellt. Informationen darüber finden sich im Eclipse Wiki [8]. Weitere aktuelle Informationen aus dem Eclipse Umfeld finden sich in Blogs die im Eclipse Planet aggregiert werden (`http://planeteclipse.org/`).

Weiterhin ist auch einige der Literatur in Buchform, die im Literaturverzeichnis dieser Arbeit verlinkt ist, sehr hilfreich. Grundsätzlich findet man jedoch die meisten Informationen in RCP-Beispielapplikationen, formloser Dokumentation und im Eclipse-(Online-

)Hilfesystem. Auch sollte man zur RCP-Entwicklung die Quellen von RCP installiert haben, sodass man bei jeder Gelegenheit nahtlos in den Quellcode von RCP wechseln kann. Die API-Dokumentation der RCP und auch GEF Klassen ist meist sehr hilfreich.

Zum Einstieg ist das RCP Tutorial von Lars Vogel empfehlenswert [5]. Zu den Grundlagen von GEF sind die Folien eines EclipseCon Tutorials interessant [9]. Tiefergehende jedoch teils veraltete Informationen bietet das IBM Redbook zu diesem Thema [10].

Einen allgemeinen Überblick und guten Enstieg in die Eclipse Plugin Entwicklung (weniger RCP) findet sich in der Seminararbeit im Kurs Software-Engineering 2011/TIT2008 von Felix Kienzle [11].

Da Eclipse RCP kein monolithisches Produkt ist sondern sich aus einzelnen Komponenten zusammensetzt findet man auch Informationen jenseits von RCP. Beispielsweise zu OSGi[12], JFace[13] oder SWT. Eine Liste von empfohlenen Büchern findet sich unter [14].

4.2 Eclipse RCP als stärkerer Bestandteil des Studiums

Im 4ten-Semester gab es eine Einführung in das Arbeiten mit Eclipse. Aus der Sicht dieser Studienarbeit lag der Focus dabei leider lediglich auf einer kurzen Einführung in die Arbeit mit den Java Development Tools (JDT). Um die Arbeit an KSM/RCP vorzubereiten könnte in dieser Vorlesung schon die Entwicklung von Eclipse-Plugins und RCP-Anwendungen besprochen werden.

Eine Möglichkeit zur Optimierung des dadurch gestiegenen Zeitbedarf könnte in der thematischen Linearisierung der Vorlesungen zu Programmiersprachen liegen. So gab es im Zuge der Vorlesungen C++, Java/Eclipse, .net/C# Themenredundanzen. Möglicherweise wäre es denkbar, anhand Java - wegen dem vglw. einfachen Aufbau dieser Programmiersprache - die Objektorientierte Programmierung zu erläutern und die vergleichsweise komplexere Einführung in C++ zu verkürzen.

Die Einführung in C und C++ könnte die *Eclipse IDE for C/C++ Developers* als primärer Entwicklungsumgebung einsetzen anstatt des bei ca. der Hälfte der Studenten nicht direkt

lauffähige *Microsoft Visual C++ Express*.

Die zusätzliche Vermittlung der .net-Umgebung mit der Sprache C# hat in meinen Augen keinen Sinn gemacht, da hier keine neuen oder andersartigen Konzepte sondern lediglich leichte Syntaxveränderungen vermittelt wurden.

Ergänzend hätte mich sehr eine Einführung in funktionale Programmierung interessiert, die könnte - bei anhaltender Eclipse RCP-Ausrichtung - mit dem Einsatz Scala erfolgen [15][16].

Als großes Open-Source Projekt könnte das Eclipse Projekt in der Vorlesung „Open-Source Systeme" näher betrachtet werden.

Die als freiwillige Veranstaltung angebotene Einführung in LaTeX könnte TeXlipse als Editor vorstellen.

5 Zusammenfassung und Ausblick

Abschlissend kann gesagt werden, dass sich der Aufwand der Einarbeitung in die Eclipse RCP Plattform gelohnt hat. Es ist empfehlenswert die KSM/Swing Entwicklung zugunsten einer intensivierten Arbeit mit Eclipse aufzugeben.

5.1 KSM als Open-Source Projekt

Bereits zu Beginn der ersten Studienarbeit wurde die Veröffentlichung des KSM Programm inklusive Quelltexte diskutiert. Zu diesem Zeitpunkt war man der Auffasung, dass KSM/Swing technisch nicht in einem veröffentlichungswüdigen Zustand ist und KSM/RCP existierte noch nicht als Prototyp. Die Veröffentlichung von KSM/Swing wird wahrscheinlich nicht mehr in dieser Studienarbeit geschehen.

Unabhängig davon erscheint eine Veröffentlichung von KSM/RCP sinnvoll. Es darf dabei jedoch nicht als klassisches Open-Source Projekt gesehen werden weil es noch nicht produktiv verwendbar ist:

> It's fairly clear that one cannot code from the ground up in bazaar style. One can test, debug and improve in bazaar style, but it would be very hard to originate a project in bazaar mode.
> ... Your nascent developer community needs to have something runnable and testable to play with.
>
> Eric S. Raymond, „Necessary Preconditions for the Bazaar Style" [17]

Die Veröffentlichung hat nicht die Absicht eine Entwicklergemeinde zu bilden, sondern eine „stabile" Heimat für KSM zu gründen. Wie in Studienarbeit 1 visualisiert wurde

[1, S. 2] war die bisherige Entwicklung von KSM eher chaotisch als zielstrebig. Dies lag vermutlich auch zum Teil daran, dass beginnende Studenten kein sauberes Projekt vorfanden sondern auf „ein SVN" was *irgendwo liegt* verwiesen wurden und dann gibts da noch so eine *CD-ROM*. Eine Übersicht über die vorhergehenden Studienarbeiten-Ausarbeitungen gab es bis dahin ebenfalls nicht. Ein im Jahrgang TIT2007 eingerichtete Redmine Projektmanagment Installation war dem it2008 Jahrgang nicht mehr zugänglich.

Die Veröffentlichung - bzw. eine sauber strukturierte Ablage - ist ein Baustein im Prozess der Zustand ändert.

In Hinblick auf eine Open-Source Entwicklung wird das private und sich als unzuverlässig herausgestellte Subversion-Repository gegen ein Git-Repository getauscht. Die Wahl für Git begründet sich durch die starke Präsenz im Open-Source Bereich, besonders auch in Eclipse nahen Projekten und dadurch, dass bereits im Kurs Software-Engineering mit Git gearbeitet wurde. Es wird die ebenfalls im Studium bereits verwendete Plattform `github.com` verwendet. Dort stehen weitere, integrierte Funktionen wie Messaging, Issue-Tracker und Wiki zur Verfügung. Diese wurden jedoch im Rahmen dieses Projektes noch nicht eingesetzt.

Die Quellen des Eclipse basierten KSM/RCP werden unter dem Github Organization-Account *dhbw-horb* (`https://dhbw-horb.github.com`) veröffentlicht bzw. hinterlegt werden. Ebenso wird ein Verweis auf die KSM-Website hinterlegt. Zukünftige Entwickler erhalten Zugriff, indem sie Mitglied dieses Organization-Account werden.

Zwar wird der Quellcode möglicherweise veröffentlicht - von Open-Source im Sinne der allgemeinen Bedeutung ist allerdings nicht zu sprechen, da keine Open-Source Lizenzierung vorgenommen wird die fremden Personen die Beteiligung erleichtern würde.

Da zur Drucklegung noch keine endgültige Entscheidung der Projektleitung über den öffentlichen Zugang zu den KSM/RCP Quellen gefallen ist, ist das KSM/RCP-Repository bis auf weiteres *private*, d.h. es kann nur von Mitgliedern der *dhbw-horb Organization* eingesehen werden.

5.2 Ausstehende Arbeiten

Mit dem GEF-basierten *Editor* lassen sich bereits Systeme modellieren. Es sind momentan aber nicht alle im Datenformat vorgesehenen und von KSM/Swing exportierten visuellen Eigenschaften implementiert.

Des Weiteren fehlt eine Eingabemöglichkeit für die bei den Simulation wichtigen Werte. Diese Eingabemöglichkeit sollte möglicherweise als seperates Plugin implem tiert werden um die Möglichkeit zu erhalten qualitative oder quantitative Prozessgrößen (QKSM) zu verwenden ohne die „Hauptapplikation" verändern zu müssen was im schlimmstenfall wieder zu einer Zerspaltung der Projekte führen könnte.

Die grafische Darstellung von Hirarchien in KSM/RCP unterscheidet sich von der Darstellung im Swing-basierten KSM. Meiner Ansicht erfordert das Thema *Hirarchien* eine radikale Lösung z.B. durch die Ersetzung von Hirarchien durch „Tags" mit denen Knoten gruppiert werden können. Probleme wie „Phantom Knoten"[18, S. 49] gibt es jedoch in KSM/RCP da die Daten im Modell auch echt hirarchisch abgelegt sind.

Im Bereich des grafischen Editors könnte möglicherweise bald unter Verwendung des Eclipse Graphiti Project welches auf GEF aufbaut einiges vereinfacht werden. Weiterhin entfällt bei Beendung der KSM/Swing Entwicklung der Grund, das Datenmodell unabhängig von Eclipse-Bibliotheken zu halten und das Datenmodell könnte möglicherweise besser mit den Eclipse Modeling Werkzeugen erstellt werden.

Mit dem *Table-Editor* Entwurf wurde gezeigt, dass die aus KSM/Swing bekannte Tabellen/Matrix mit Eclipse-RCP umsetzbar ist. Der Prototyp ist lediglich in der Lage die Eigenschaft „Beschreibung" (`visual.caption`) zu editieren. Es steht aus, dem Benutzer zu ermöglichen alle Eigenschaften, die auch durch weitere Plugins dynamisch definiert werden können, zu editieren. Bisher unterstützt der Table-Editor nur den Datentyp `Node` (KSM-Knoten). Sowohl `NodeGroup` („Hirarchie") als auch `Connection` sollten unterstützt werden um den vom KSM/Swing gewohnten Funktionsumfang zu bieten.

Der Prototyp des *Simulators* zeigt, dass sich Darstellungen als Line-Chart live mit den KSM-Daten im Editor verknüpfen lassen. Die eigentliche Funktionalität des Simulierens

ist nicht implementiert.

In der *Datenmodell* Library `ksm-model` sollte es möglich sein, Vererbung zu Nutzen um die Model-Klassen für den Einsatzbereich zu erweitern. Hierzu müsste zur Instantierung neuer Model-Objekte jedoch ein Factory-Pattern eingesetzt werden was nicht der Fall ist.

Auch in der Datenmodell Library wurde der Ansatz gewählt, dass sich auf alle Objekte Listener registrieren lassen. Dies bedeutet jedoch auch, dass um alle Änderungen zu registrieren ein Listener erst auf jedes Objekt in der Modell Objekt-Hierarchie registriert werden muss. Vermutlich wäre es klüger gewesen nur ein Listener global für das gesamte Modell zu registrieren und die Zuordnung der Events zu Knoten, Verbindungen usw. im Event mitzugeben.

Anhang 1. Dokumentation Datenmodell

Contents

1 Einleitung

Mit der Entwicklung einer KSM Anwendung auf Eclipse-RCP Basis war es nötig ein Datenmodell zu entwerfen welches zum MVC-ähnlichen Muster des Eclipse Graphical Editor Framework (GEF) kompatibel ist.

Grundsätzlich lassen sich alle Datenmodelle mit GEF abbilden, jedoch existierte in KSM/Swing nur ein stark mit der GUI und Logik verflochtenes und fehlerhaftes Datenmodell, dessen Serialisierung mit einer XML-Bibliothek erfolgte die nicht mehr weiter verwendet werden soll.

Da es nicht beabsichtigt ist die KSM/Swing Anwendung auf kurze Zeit abzulösen liegt es nahe, dass das Datenmodell universell einsetzbar sein sollte. Mit der Umstellung von KSM/Swing auf ein neues Datenmodell ist einerseits die Daten-Kompatibiltät zwischen KSM/Swing und KSM/RCP gegeben und das Ziel die XML-Serialisierung in KSM/Swing zu überarbeiten kann einfacher erreicht werden.

Dieses Dokument beschreibt die Implementation einer API zum Zugriff auf das in XML-Schema beschriebene Datenformat.

Dieses Datenformat erlaubt es darüberhinaus freie Felder zu definieren welche ebenfalls in diesem Dokument spezifiert werden.

2 Das XML-Schema

Die Namespace URL für das XML-Schema lautet:

`http://www.ba-horb.de/~ksm/xml/ksm-1`

die Schemadatei ist abrufbar unter:

`http://www.ba-horb.de/~ksm/xml/ksm-1.xsd`

Das XML-Schema ist angereichert um JAXB-Annotationen die den XML-Schema-Java-Compiler bei der Klassenerzeugung steuert.

3 Typen des Datenmodell

Die Bibltiohekt besteht aus einem Package in dem Interfaces die API-Beschreiben und einem Package mit Implementationen dieser Interfaces die jedoch für den Benutzer unsichtbar und daher austauschbar sind.

Das folgende Diagram zeigt das Interface das dem Benutzer zur Verfügung steht:

Ein Diagramm wird erstellt indem ein neues (*createEmptyKSM()*) erstellt wird oder eines geladen wird (*loadKSM()*). Die weitere Navigation erfolgt durch Absteigen in dem Objekt-Baum der durch NodeGroup- und schliesslich Node-Objekte gebildet wird.

3.1 Datentypen der Eigenschaften

Im Modell gibt es die Datentypen KSM, NodeGroup, Node und Connection (siehe Klassen-Diagramm). Zusätzlich gibt es die Klasse Properties mit der an die Datentypen zusätzliche Eigenschaften mit einem Text-Bezeichner angehängt werden können.

Für die Eigenschaften stehen folgende Typen zur Verfügung: *'string*', *'integer*', *'boolean*', *'integerList*', *'decimalList*', *'stringList*'. Der Datentyp erschliesst sich aus dem Namen.

Prinzipiell kann jeder Bezeichner für jeden Datentyp einmal verwendet werden, ein Bezeichner sollte jedoch wegen der Übersichtlichkeit nur einmal verwendet werden.

4 Verwendung der ChangeListener

Die Klassen *NodeGroup*, *Node* und *Properties* implementieren Change-Listener auf die sich Listener-Klassen registrieren können um über Änderungen am Datenmodell informiert zu werden.

Dies wird beispielsweise benötigt, wenn zwei Programmteile wie ein Eigenschaften-Editor in Tabellenform und ein grafischer Editor voneinander unabhängig auf das Datenmodell zugreifen und beispielsweise die Farbe eines Knoten ändern.

Die Spalte *Index* zeigt an ob das Event eine Index-Eigenschaft hat die andeutet welches Element in einer Liste geändert wurde.

Die Spalte *Version* zeigt an, ab welcher Version des Datenformat (=Version dieses Dokumentes, Version Attribut in <ksm> Element) dieser Event unterstützt wird.

Table 1: *NodeGroup*-Events

Property-Name	**Version**	**Index?**	**Beschreibung**
nodes	1+	✓	Eine *Node* wurde dieser *NodeGroup* hinzugefügt oder entfernt

Table 2: *Node*-Events

Property-Name	**Version**	**Index?**	**Beschreibung**
connections	1+	✓	Eine *Connection* wurde erstellt oder gelöscht

Table 3: *Properties*-Events

Property-Name	**Version**	**Index?**	**Beschreibung**
string: *X*	1+	✗	Eine Zeichenketten Eigenschaft mit Name *X* wurde geändert
decimal: *X*	1+	✗	Eine Fliesskommazahl Eigenschaft mit Name *X* wurde geändert
integer: *X*	1+	✗	Eine ganzzahlige Eigenschaft mit Name *X* wurde geändert
boolean: *X*	1+	✗	Eine boolsche Eigenschaft mit Name *X* wurde geändert
integerList: *X*	1+	✓	Eine Ganzzahl Liste mit Name *X* wurde manipuliert
decimalList: *X*	1+	✓	Eine Fliesskommazahl Liste mit Name *X* wurde manipuliert
stringList: *X*	1+	✓	Eine Zeichenketten Liste mit Name *X* wurde manipuliert

5 Eigenschaften (Properties)

Allen Elementen im Datenmodell lassen sich dynamische, dass heist nicht in einem Schema festgelegte, Eigenschaften zuwiesen. Dies hat zur Folge, dass eine Anwendung sowohl den Fall handhaben muss, dass eine erwartete Eigenschaft nicht vorhanden ist als auch, dass Eigenschaften vorhanden sind die unbekannt sind und ignoriert werden müssen. Dazu steht der Anwendung jedoch das Attribut *Version* im Schema (siehe KSM#getVersion()) zur Verfügung, welches auf eine Version von diesem Dokument zeigt.

Dieser Ansatz wurde gewählt, da sich gezeigt hat, dass die Studienarbeiten im KSM-Projekt einen begrenzten Fokus haben und es daher für einen einzelnen Studenten schwer möglich ist, alle benötigten Datenfelder zu definieren. Das bisherige KSM-Datenformat handhabt dies, indem das XML-Schema beliebig verändert wurde und damit sinnlos wurde. Da dies unvermeidbar ist wird es mit diesem Ansatz aktiv unterstützt.

Eine alternative Herangehensweise wäre die Verwendung von verschiedenen XML-Schemas gewesen wobei mit jeder Erweiterung ein zusätzlicher Namensraum eingeführt wird. Dies schien jedoch sehr viel umständlicher und unnötig kompliziert.

5.1 Eigenschaften von Modellen (KSM)

Die Spalte *Typ* zeigt den Datentyp der Eigenschaft an. Die Spalte *Schlüssel* den Bezeichner welcher in Kombination mit dem Typ eindeutig ist.

Die Spalte *Version* zeigt an, ab welcher Version des Datenformat (=Version dieses Dokumentes, Version Attribut in <ksm> Element) dieser Event unterstützt wird.

Table 4: KSM Properties

Typ	**Schlüssel**	**Version**	**Beschreibung**
nichts	ist	definiert	-

5.2 Eigenschaften von Knoten (Node)

Table 5: Node Properties

Typ	**Schlüssel**	**Ver.**	**Beschreibung**
string	visual.caption	1+	Titel der Node im Editor.
string	visual.color	1+	Farbliches Merkmal der Node Hexadezimal im 8-Bit RGB Format wie folgt: `#RRGGBB`.
integer	visual.location.x	1+	X-Position relativ zur übergeordneten NodeGroup
integer	visual.location.y	1+	Y-Position relativ zur übergeordneten NodeGroup
decimal	data.user_value	1+	User Value (?)

Table 5: (continued)

Typ	Schlüssel	Ver.	Beschreibung
decimal	data.min_value	1+	Minimal Value (?)
decimal	data.max_value	1+	Maximal Value (?)
decimal	data.extern	1+	Extern Value (?)

Als Erbe aus dem KSM/Swing Projekt kann *'visual.color'* die folgenden Werte annehmen, diese sollen auf den folgenden RGB-Wert übertragen werden:

- `White` → `#ffffff`
- `Light Yellow` → `#faffa2`
- `Medium Yellow` → `#f4ff4b`
- `Yellow` → `#edfc00`
- `Light Blue` → `#d4d5e9`
- `Medium Blue` → `#7678ff`
- `Blue` → `#0002f8`
- `Light Green` → `#c8f8c9`
- `Medium Green` → `#7afa7e`
- `Green` → `#1af520`
- `Light Red` → `#fdcccc`
- `Medium Red` → `#f95959`
- `Red` → `#f62020`

5.3 Eigenschaften von Verbindungen (Connection)

Table 6: Connection Properties

Typ	Schlüssel	Ver.	Beschreibung
string	visual.caption	1+	Titel der Connection im Editor.
string	visual.color	1+	Farbliches Merkmal der Connection
string	data.functionType	1+	Ein Funktionstyp von: *straight-line*, *individual*, *parable*, *parabolic-sections*.
decimalList	data.function	1+	KSM-Simulator Funktionsparameter dieses Knoten

'data.function' enthält eine Liste von Argumenten für die verwendete, durch *'data.functionType'* festgelegte Funktion.

5.4 Eigenschaften von Gruppen (NodeGroup)

Table 7: NodeGroup Properties

Typ	**Schlüssel**	**Ver.**	**Beschreibung**
string	visual.caption	1+	Titel der Group im Editor.
string	visual.color	1+	Farbliches Merkmal der Group
integer	visual.location.x	1+	X-Position relativ zu übergeordneten NodeGroup
integer	visual.location.y	1+	Y-Position relativ zu übergeordneten NodeGroup

6 Dokumentation der Implementierung

Das am Anfang vorgestellte Objektmodell wird durch Interfaces im Package *de.dhbw.horb.ksm. xmlschema.api* umgesetzt.

Eine Implementierung dieser Interfaces findet sich im Package *de.dhbw.horb.ksm. xmlschema.imp* Von dieser Implementierung ist für äusseren Zugriff nur die Klasse *KSMFactory* vorgesehen, welche Methoden zum Laden und Speichern von KSM-Modellen zur Verfügung stellt.

Die Implementierung benutzt vom XML-Schema-Compiler (xjc) von JAXB generierte Klassen die im Package *de.dhbw.horb.ksm. xmlschema.generated* liegen. Das generieren wird vom Ant-Task *compile-xjc* und Annotationen im Schema gesteuert.

Der Zugriff auf das geladene Modell erfolgt auschliesslich über die in den Interfaces vorgesehen Methoden, ein direkter Zugriff ist nicht möglich.

Neben dem Quellcode in src/ gibt es im Projektverzeichniss noch das Verzeichnis test/ welches JUnit-4 Tests enthält die die Implementierung nahezu 100% abdecken. Die Tests sind dabei zum Teil im Stil des Behavior Driven Development (BDD) geschrieben unter Zuhilfename der Bibliothek mockito.

7 Vorgehensweise bei Änderungen

1. Eintragung der Änderung in diesem Dokument
2. Erhöhen der Versionszahl
3. Erstellen eines Eintrags in der Revisions Historie in diesem Dokument
4. Anpassen der Versionszahl in build.xml *project.version*
5. Erstellen einer HTML- und PDF Version von diesem Dokument (asciidoc/a2x).

8 Revisions Historie

Table 8: Revisions

Ver.	**Datum**	**Person**	**Änderung**
1	2011-03-24	Yves Fischer	Beginn der Historie, KSM Version 1
1	2011-04-26	Fischer, Dreher	Erläuterung Datentypen. Festlegung Funktionsname/Parameter von Connection's.

Anhang 2. Neues XML-Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.ba-horb.de/~ksm/xml/ksm-1"
  xmlns:tns="http://www.ba-horb.de/~ksm/xml/ksm-1" elementFormDefault="qualified"
  xmlns:jxb="http://java.sun.com/xml/ns/jaxb" jxb:version="1.0">

  <!-- Wurzelelement eines KSM Dokument -->
  <element name="ksm">
    <complexType>
      <annotation>
        <appinfo>
          <jxb:class name="XKSM" />
        </appinfo>
      </annotation>
      <sequence>
        <element name="nodegroup" type="tns:NODEGROUP_TYPE"
          minOccurs="1" maxOccurs="1">
        </element>
        <element name="properties" type="tns:PROPERTIES_TYPE"
          minOccurs="0" maxOccurs="1"></element>
      </sequence>
      <attribute name="version" type="string" use="required" />
    </complexType>
  </element>

  <complexType name="PROPERTY_BASE_TYPE">
    <annotation>
      <appinfo>
        <jxb:class name="XPropertyBase" />
      </appinfo>
    </annotation>
    <simpleContent>
      <extension base="anySimpleType">
        <attribute name="name" use="required" />
      </extension>
    </simpleContent>
  </complexType>

  <complexType name="PROPERTY_STRING_TYPE">
```

```
  <annotation>
    <appinfo>
      <jxb:class name="XPropertyString" />
    </appinfo>
  </annotation>
  <simpleContent>
    <extension base="string">
      <attribute name="name" use="required" />
    </extension>
  </simpleContent>
</complexType>

<complexType name="PROPERTY_BOOLEAN_TYPE">
  <annotation>
    <appinfo>
      <jxb:class name="XPropertyBoolean" />
    </appinfo>
  </annotation>
  <simpleContent>
    <extension base="boolean">
      <attribute name="name" use="required" />
    </extension>
  </simpleContent>
</complexType>

<complexType name="PROPERTY_DECIMAL_TYPE">
  <annotation>
    <appinfo>
      <jxb:class name="XPropertyDecimal" />
    </appinfo>
  </annotation>
  <simpleContent>
    <extension base="decimal">
      <attribute name="name" use="required" />
    </extension>
  </simpleContent>
</complexType>

<complexType name="PROPERTY_INTEGER_TYPE">
  <annotation>
    <appinfo>
      <jxb:class name="XPropertyInteger" />
    </appinfo>
  </annotation>
  <simpleContent>
    <extension base="integer">
      <attribute name="name" use="required" />
    </extension>
  </simpleContent>
</complexType>
```

```
<complexType name="PROPERTY_LIST_BASE_TYPE">
  <annotation>
    <appinfo>
      <jxb:class name="XPropertyBaseList" />
    </appinfo>
  </annotation>
  <sequence>
  </sequence>
  <attribute name="name" use="required"></attribute>
</complexType>

<complexType name="PROPERTY_STRING_LIST_TYPE">
  <annotation>
    <appinfo>
      <jxb:class name="XPropertyStringList" />
    </appinfo>
  </annotation>
  <complexContent>
    <extension base="tns:PROPERTY_LIST_BASE_TYPE">
      <sequence>
        <element name="value" type="string" minOccurs="0"
          maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="PROPERTY_DECIMAL_LIST_TYPE">
  <annotation>
    <appinfo>
      <jxb:class name="XPropertyDecimalList" />
    </appinfo>
  </annotation>
  <complexContent>
    <extension base="tns:PROPERTY_LIST_BASE_TYPE">
      <sequence>
        <element name="value" type="decimal" minOccurs="0"
          maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="PROPERTY_INTEGER_LIST_TYPE">
  <annotation>
    <appinfo>
      <jxb:class name="XPropertyIntegerList" />
    </appinfo>
  </annotation>
  <complexContent>
```

```
    <extension base="tns:PROPERTY_LIST_BASE_TYPE">
      <sequence>
        <element name="value" type="integer" minOccurs="0"
          maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>

<!-- EinelListe von Eigenschaften -->
<complexType name="PROPERTIES_TYPE">
  <annotation>
    <appinfo>
      <jxb:class name="XProperties" />
    </appinfo>
  </annotation>
  <sequence>
    <element name="string" type="tns:PROPERTY_STRING_TYPE"
      minOccurs="0" maxOccurs="unbounded" />
    <element name="decimal" type="tns:PROPERTY_DECIMAL_TYPE"
      minOccurs="0" maxOccurs="unbounded" />
    <element name="integer" type="tns:PROPERTY_INTEGER_TYPE"
      minOccurs="0" maxOccurs="unbounded" />
    <element name="boolean" type="tns:PROPERTY_BOOLEAN_TYPE"
      minOccurs="0" maxOccurs="unbounded" />
    <element name="stringList" type="tns:PROPERTY_STRING_LIST_TYPE"
      minOccurs="0" maxOccurs="unbounded" />
    <element name="decimalList" type="tns:PROPERTY_DECIMAL_LIST_TYPE"
      minOccurs="0" maxOccurs="unbounded" />
    <element name="integerList" type="tns:PROPERTY_INTEGER_LIST_TYPE"
      minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>

<!-- KSM Node (Knoten), hat Eigenschaften und Verbindungen zu anderen Knoten -->
<complexType name="NODE_TYPE">
  <annotation>
    <appinfo>
      <jxb:class name="XNode" />
    </appinfo>
  </annotation>
  <sequence>
    <element name="properties" type="tns:PROPERTIES_TYPE"
      minOccurs="1" maxOccurs="1" />
    <element name="connections" type="tns:CONNECTIONS_TYPE"
      minOccurs="0" maxOccurs="1" />
  </sequence>
  <attribute name="id" type="tns:NODE_ID_TYPE" use="required" />
</complexType>
```

```
  <!-- Eine NodeGroup fasst mehrere Knoten zusammen, auch bekannt als Hirachien -->
  <complexType name="NODEGROUP_TYPE">
    <annotation>
      <appinfo>
        <jxb:class name="XNodeGroup" />
      </appinfo>
    </annotation>
    <sequence>
      <element name="node" type="tns:NODE_TYPE" minOccurs="0"
        maxOccurs="unbounded" />
      <element name="nodegroup" type="tns:NODEGROUP_TYPE"
        minOccurs="0" maxOccurs="unbounded"></element>
      <element name="properties" type="tns:PROPERTIES_TYPE"
        minOccurs="1" maxOccurs="1" />
    </sequence>
    <attribute name="id" type="tns:NODE_ID_TYPE" use="required" />
  </complexType>

  <!-- Connections. Eine Node eine beliebige Anzahl gerichteter Verbindungen
    zu einer anderen Node haben -->
  <complexType name="CONNECTIONS_TYPE">
    <annotation>
      <appinfo>
        <jxb:class name="XConnections" />
      </appinfo>
    </annotation>
    <sequence>
      <element name="connection" type="tns:CONNECTION_TYPE"
        minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>

  <!-- Eine gerichtete Verbindung zu einer anderen Node. -->
  <complexType name="CONNECTION_TYPE">
    <annotation>
      <appinfo>
        <jxb:class name="XConnection" />
      </appinfo>
    </annotation>
    <sequence>
      <element name="properties" type="tns:PROPERTIES_TYPE"
        minOccurs="1" maxOccurs="1" />
    </sequence>
    <attribute name="to" type="tns:NODE_ID_TYPE" />
  </complexType>

  <simpleType name="NODE_ID_TYPE">
    <restriction base="string"></restriction>
  </simpleType>
</schema>
```

Anhang 3. Inhalt der beigelegten CD

`ausarbeitung/` Schriftliche Arbeit im PDF.

`ksm-rcp/...` Aktueller Stand des Projektrepositories. Siehe Kapitel 3 ab Seite 11.

`ksm` Kopie des aktullen Stand im gemeinsamen Subversion Repository.

Anhang 4. Abbildungsverzeichnis

Anhang 5. Literatur

[1] Yves Fischer. KSM - Eclipse RCP. Studienarbeit an der DHBW Stuttgart Campus Horb, 2010.

[2] Friedhelm Wolf. Implementierung eines Kybernetischen System Modells. Studienarbeit an der BA-Horb, 2005.

[3] Tobias Dreher. Refactoring und Redesign. Studienarbeit an der DHBW Stuttgart Campus Horb, 2010.

[4] Tobias Dreher. Refactoring und Redesign (2). Studienarbeit an der DHBW Stuttgart Campus Horb, 2011.

[5] Lars Vogel. Eclipse RCP Tutorial, 2011. Version 5.7, http://www.vogella.de/articles/EclipseRCP/article.html.

[6] *Create an Eclipse-based application using the Graphical Editing Framework.*

[7] Christian Riess. Kybernetisches Systemmodell KSM 4. Studienarbeit an der BA-Horb, 2003.

[8] Eclipse Wiki. http://wiki.eclipse.org/Main_Page.

[9] Randy Hudson and Pratik Shah. *GEF in Depth.* IBM Rational Software, 2005. http://www.eclipsecon.org/2005/presentations/EclipseCon2005_23_GEF_Tutorial_Final.ppt.

[10] William Moore, David Dean, Anna Gerber, Gunnar Wagenknecht, and Philippe Vanderheyden. *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework.* An IBM Redbooks publication, 2004. http:

//www.redbooks.ibm.com/abstracts/sg246302.html.

[11] Felix Kienzle. Plugin Entwicklung: Eclipse Plugin Entwicklung. Seminararbeit Software-Engineering bei Prof. Dr. phil. van Hoof an der DHBW Stuttgart Campus Horb, 2011.

[12] Neil Bartlett. *OSGi in Practice.* Bartlett, Neil, 2010. http://njbartlett.name/osgibook.html.

[13] Matthew Scarpino, Stephen Holder, Stanford Ng, and Laurent Mihalkovic. *SWT/JFace In Action.* Manning, 2005.

[14] Chris Aniszczyk, EMC. *Recommended Eclipse reading list.* IBM developerWorks, 2006. https://www.ibm.com/developerworks/library/os-ecl-read/.

[15] Scala Bundle. http://wiki.eclipse.org/Scala_Bundle.

[16] Using Scala to Create Eclipse RCP Applications. http://www.coconut-palm-software.com/the_new_visual_editor/doku.php?id=blog:using_scala_to_create_eclipse_rcp_applications.

[17] Eric Steven Raymond. The Cathedral and the Bazaar, 2002. http://www.catb.org/~esr/writings/homesteading/cathedral-bazaar/index.html.

[18] Thomas Pustelnik. Implementierung von Hirarchien in einem KSM. Studienarbeit an der DHBW Stuttgart Campus Horb, 2010.

[19] Berthold Daum. *Java-Entwicklung mit Eclipse 3.1.* dpunkt.verlag, 2005.

[20] Scott Delap. *Understanding how Eclipse plug-ins work with OSGi.* IBM developerWorks, 2006. http://www.ibm.com/developerworks/library/os-ecl-osgi/index.html.

[21] Yannick Saillet. *Migrate your Swing application to SWT.* IBM developerWorks, 2004. http://www.ibm.com/developerworks/java/tutorials/

j-swing2swt/.

[22] David Gallardo. *Developing Eclipse plug-ins*. IBM developerWorks, 2002. http://www.ibm.com/developerworks/opensource/library/os-ecplug/.

[23] Berthold Daum. *Das Eclipse-Codebuch*. dpunkt.verlag, 2006. aus Bibliothek DHBW/Horb.

[24] Gamma, Nackman, and Wiegand. *eclipse - Building Commercial-Quality Plug-ins*. Addison-Wesley, 2006. aus Bibliothek DHBW/Horb.

[25] Sippel, Jastram, Bendisposto. *Eclipse Rich Client Platform*. entwickler.press, 2009. aus Bibliothek DHBW/Horb.