Erstellung eines Prototypen zur Evaluierung der durch das Graphical Editing Framework bereitgestellten Funktionen

Entstanden im Rahmen eines Softwarprojekts an der HdM-Stuttgart

betreut von Herrn Prof. Walter Kriha

angeregt durch die Ebdac Software GmbH

com.ebdac.gef.dbmodeller
Version 0.9 (Stand 28. Januar 2006)

Andreas Seelig

Student Medieninformatik
Matrikelnr. 12 6 35
as44@hdm-stuttgart.de

Inhaltsverzeichnis

Inhaltsverzeichnis 1
1. Das Graphical Editing Framework 2
 1.1. Draw2D 2
 1.2. GEF 3
2. Das Projekt 5
 2.1. Motivation 5
 2.2. Zeitliche Gliederung des Projektes 5
 2.3. Einarbeitung 6
 2.4. Portieren des Codes (von SWT nach GEF) 6
 2.5. Das Modell 7
 2.6. Der Controler 7
  2.6.1. EditPart 8
 2.7. Die View 11
  2.7.1. IFigure 11
 2.8. Anschluß der Outline und der PropertyView 12
 2.9. Lessens learned 12
 2.10. Fazit 13
3. Ausblick 14
4. Quellen 15
5. Der DB-Modeller 16
 5.1. Überblick 16
 5.2. DB-Modeller als Eclipse Plug-in 16
 5.3. Die DatabeseModule.dtd 16
  5.3.1. Tables 17
  5.3.2. Views 17
  5.3.3. Procedures 17
  5.3.4. Restrictions 17
  5.3.5. Projections 17
  5.3.6. Die vollständige DatabaseModule.dtd 18
 5.4. Die Installation des Plug-ins 20
 5.5. Dataiformate 20
  5.5.1. Das .xml-File 20
  5.5.2. Das .dbm-File 20
 5.6 Editieren 21
  5.6.1. Neues Modul anlegen 21
  5.6.2. Modelling -View 22
 5.7. Edit-Views 24
  5.7.1. Anzeigen der Edit-Views 24
  5.7.2. Model View 25
  5.7.3. Table View 25
 5.8. Tables und Views 26
  5.8.1. Primary-Key 26
  4.8.2. Foreign-Key 27
 4.9. Outline 28
6. Anlagen: 29

1. Das Graphical Editing Framework

Das Graphical Editing Framework (GEF) dient als Grundlage zur Erstellung von maßgeschneiderten Editoren welche mit den Möglichkeiten der Eclipse-Plattform nur schwer zu realisieren wären. Grundlegende Anforderungen an einen graphischen Editor wie beispielsweise

- Inhalt der Daten und ihrer Struktur in einer für den Menschen “rel. intuitiven” Form darstellen (alles was sich in graphenähnlicher Form darstellen lässt)
- Daten inhaltlich verändern
- Beziehungen der Daten untereinander verändern
- Einhaltung der Strukturierungsmöglichkeiten der Daten beim Erzeugen/Löschen/Verändern überprüfen
- Redo/Undo
- Copy/Past
- DragAndDrop
- Drucken
- ...

können durch den Einsatz von GEF leicht realisiert werden. GEF folgt dabei streng dem MVC-Paradigma.

GEF als Komponente teilt sich in zwei Plug-ins:

- Draw2d (`org.eclipse.draw2d`) - das lightweight toolkit um auf dem SWT Canvas zu zeichnen und layouten
- GEF (`org.eclipse.gef`) - ein MVC Framework welches auf Draw2d aufbaut und Interaktionen mit dem Model ermöglicht

1.1. Draw2D

Draw2d ist ein lightweight toolkit bestehend aus graphischen Komponenten, sogenannten ***figures***. Lightweight heist, daß eine Figur ein einfaches Java Object ohne entsprechenden Resourcen im Betriebssystem ist. Figures können über eine Eltern-Kind-Beziehung zusammenhängen. Jede Figur ist durch seine ***bounds*** (x- und y-Koordinaten sowie die Höhe und Breite) beschrieben innerhalb welcher sie selbst und die enthaltenen Figuren angezeigt werden. Ein Layoutmanager ist für die Anordnung der enthaltenen Figuren verantworlich.

Draw2D kann auch unhabhängig von GEF verwendet werden.

Ein `LightweightSystem` verbindet die Figuren mit einem SWT Canvas. Das lightweight system empfängt fast alle SWT-Events und leitet diese an einen `EventDispatcher weiter`, welcher diese in Events der dazugehörigen Figur umwandelt.

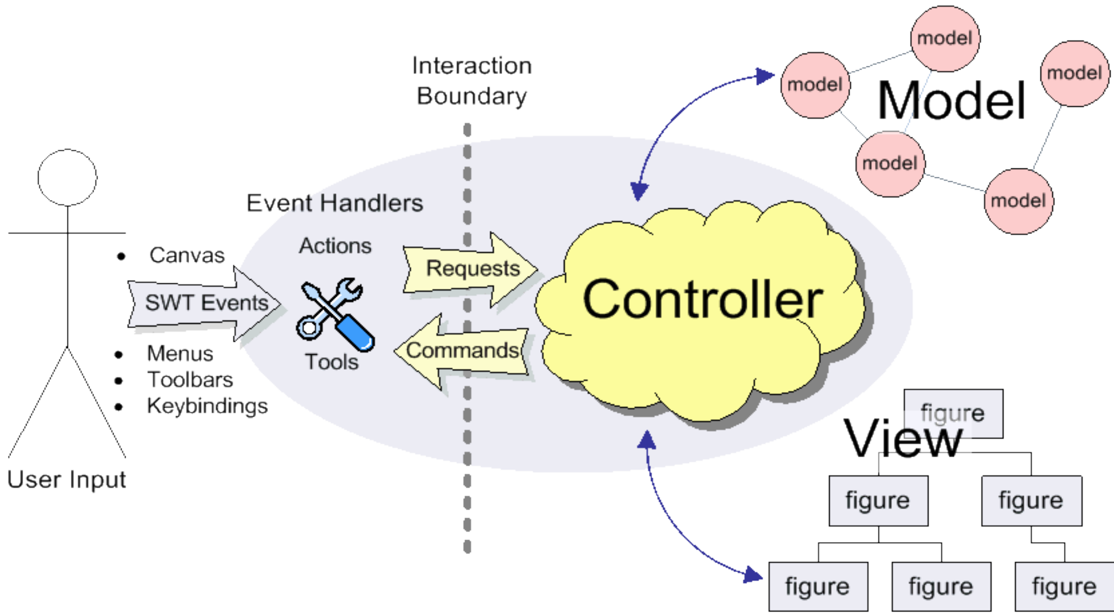
Paint events werden an den `UpdateManager weitergeleitet`, welcher das Malen und Anordnen der Figuren übernimmt. Figuren können in ihrer Erscheinung und in ihrer Größe verändert werden und der update manager stellt sicher daß nur ein layout vorhanden ist und ein repaint in der Region erfolgt welches verändert wurde.

1.2. GEF

Während Draw2D für das effiziente Darstellen der Figuren verantwortlich ist, fügt das GEF-Plugin Funktionalität zum editieren hinzu. Sinn und Zweck dieses Frameworks ist es:

1. Die grafische Darstellung jeglicher Modelle mit Hilfe von Draw2D Figures
2. Interaktionen des Benutzer mit dem Modell ermöglichen (z.B. Tastatureingaben, Mouseevents oder Interaktionen über die Workbench)

Das untenstehende Diagram zeigt eine high-level Ansicht von GEF. Ganz grob kann GEF als der violett hinterlegte Teil betrachtet werden. Zu implementieren ist im wesentlichen der Teil rechts der InteractionBoundery. Das Framework stellt die Verbindung vom Model der Applikation und der Anzeige her. Links der InteractionBoundery im Bild unten, zeigt den Teil von GEF in welchem verschiedene Events welche durch den Benutzer hervorgerufen wurden abgefangen werden und in entsprechende Requests umgewandelt werden. Requests und Commands werden verwendet um Interaktionen und ihre Auswirkung auf das Model zu kapseln.


Interaction
Boundary
model
model
model
model
model
Model
Event Handlers
Actions
Canvas
SWT Events
Requests
Controller
Commands
Menus
Toolbars
Keybindings
Tools
figure
View
figure
figure
figure
figure
figure
User Input

2. Das Projekt

In diesem Projekt ging es nicht um die Neuerstellung von Software, sondern, wie es wohl häufig der Fall ist, um das Verändern einer bereits bestehenden Struktur. Hierbei sind andere Umstände zu berücksichtigen als bei der Neuerstellung. So war hier z.B. die Outline wie auch die „PropertyPanes“ in welchen die eigenschaften der Objekte detailiert angegeben werden bereits gegeben. Dadurch konzentrierte sich die Umgestaltung im Wesentlkichen auf den Editor in welchem die Modelobjekte angezeigt werden und den entsprechenden Controlern sowie den adäquten Anschluss der beiden Komponenten Ouline und PropertyPane. Eine Analyse- und Degnphase konnte daher übersprungen werden.
Im Folgenden werde ich einige Punkte aus meiner Arbeit herausgreifen und kurz darauf eingehen. Da sowohl der Platz als auch die Zeit begrenzt ist, werde ich bei weitem nicht auf alles eingehen können da das Graphical Editing Framework als Komponente derart umfangreich ist, daß sich darüber sicherlich ein in der Dicke ansehnliches Buch schreiben liese (was ich leider keines bei meiner Recherche gefunden habe!)

2.1. Motivation

In meinem ersten Praxissemester erstellte ich ein Plugin für die IDE Eclipse zu graphischen Datenbankmodellierung bei der Ebdac Software GmbH. Hierbei verwendete ich zur Anzeige die in Eclipse eingebundene Grafikbibliothek SWT (und JFace). Dabei mussten alle gewünschte Zeichenroutienen implementiert werden welche zum Teil recht umfangreich und komplex waren. Dies machte den Quellcode relativ schwer lesbar. Zu diesem Zeitpunkt war für uns das Graphical Editing Framework nicht bekannt.
Als ich nun gefragt wurde ob ich denn nicht versuchen möchte den Code des Plug-ins auf GEF zu portieren, hielt ich dies für eine sehr interessante Aufgabe, da GEF nach dem ersten Eindruck genau für diesen Fall gedacht ist: ein Datenmodell graphisch darzustellen und mit diesem zu interagieren.
An dieser Stelle sei der Ebdac Software GmbH herzlich für die Möglichkeit und die exelente Betreung gedankt!

2.2. Zeitliche Gliederung des Projektes

Da ich mir die Vorlesungszeit etwas entspannen wollte, begann ich mit dem Projekt bereits in den (Sommer-) Semesterferien. Am Ende dieser hatte ich über das doppelte an Zeit der dafür vorgesehenen sechs SWS bereits investiert und es blieben mir auch für die Zeit unter dem Semester noch genügend Fragen die noch zu beantworten waren. Grob teilte sich das Projekt in die folgenden Phasen:

Einlesen in die Documentation von GEF (und Draw2D)

Nachvollziehen von Beispielen/Tutorials

Portieren des Codes von SWT nach GEF

Präsentation

Dokumentation

2.3. Einarbeitung

Nun ging es erst einmal darum, ein grobes Verständnis für die Arbeitsweise von GEF zu bekommen. Hierzu versuchte ich Literatur zu finden was sich als relativ schwer erwies (ich fand kein Buch welches sich mit diesem Famework ausgiebig beschäftigt!) so dass ich mich erst mal auf die Online Help und das Tutorial von Randy Hudson konzentrierte. Später fand ich dann noch einen für mich sehr hilfreichen Artikel von Phil Zoio welcher sich mit der Erstellung eines Datenbank Schema Diagram Editor befasst. Im Oktober 2005 wurde die Onlinedokumentation neu veröffentlicht welche eine deutlich bessere Einführung in das Framework bietet. Für mich war das zwar schon relativ spät aber noch nicht zu spät.

2.4. Portieren des Codes (von SWT nach GEF)

Nach dem ich mich etwas kundig gemacht hatte über den Aufbau von GEF, fing ich nun an das Beispielplugin welches in dem Artikel „Building a Database Scheme Diagram Editor“ beschrieben wird, an die von mir gewünschte Funktionalität anzupassen. Dies erleichterte mir den Einstieg da ich nicht völlig bei Null anfangen musste. Das Beispielplugin bot schon einiges an Funktionalität welche ich auch benötigte und hatte ein ähnliches Datenmodel wie ich. So versuchte ich durch den Code zu navigieren und flickte es an denen Stellen an denen ich annahm sie seien für das Verantwortlich was ich wollte. Dies funktionierte zu Beginn recht gut da ich immer etwas angezeigt bekam. Je länger ich mich nun auf diese Weise vorarbeitete, desto größer wurde aber auch das Durcheinander von für mich nützlichem Code und Code der eben das Beispielplugin am laufen hielt. Ich übernahm eine Struktur, die gar nicht exakt für mein Problem gedacht war und es kostete mich relativ viel Zeit und Geduld, den brauchbaren Code von den Code der einfach aus dem Beispiel her noch drin war zu befreien. Dies ist bis heute noch nicht vollständig abgeschlossen da ich mich erst noch um andere Fragen kümmern wollte.

Im Folgenden möchte ich nun einige Punkte herausstellen, die mir bei der Arbeit mit GEF als wichtig erschienen:

2.5. Das Modell

GEF stellt an das Modell welches dargestellt und editiert werden soll keinerlei Anforderungen! Sinnvoll ist jedoch irgend eine Art der Notification um dass sich das Model bei einer Änderung bemerkbar machen kann, der Controller darauf reagieren und die Änderung angezeigt werden kann. Dies kann über im EMF (Eclipse Modeling Framework) enthaltene Mechanismen erfolgen oder z.B über den **PropertyChangeSupport** der Java Beans.
In meinem Fall war es Möglich eine gemeinsame Superklasse an die Spitze der Vererbungspyramiede zu setzen in welcher ich eine Compositebeziehung zu einem PropertyChangeSupport implementierte und diese Superclass (PropertyAwareObject) mit den drei Methoden `addPropertyChangeListener(PropertyChangeListener l)`, `removePropertyChangeListener(PropertyChangeListener l)` und `firePropertyChange(String prop, Object old, Object newValue)` zu „dekorieren". Indiesen Methoden werden die Parameter an das referenzierte PropertyChangeSupport-Objekt delegiert. Dadurch kann sich der Controler als PropertyChangeListener bei dem Modelobject regisirieren. Nun war nur in den Methoden in denen der Zustand des Modelobjektes verändert wird der Aufruf `firePropertyChange(String prop, Object old, Object newValue)` erfolgen.
Eine kleine Änderung an den ForeignKeys war noch nötig: bisher hatte das ForeignKey Object nur eine Referenz auf die Column mit dem Primary Key auf welche er zeigt, nicht jedoch eine Referenz auf die Column von der er kommt. Dies war noch zu ergänzen um das entsprechende Controllerobject in GEF zu bedienen.
Das wars.

Eine detailierte Ansicht des gesamten Models findet sich in der beiligenden Javadoc.

2.6. Der Controler

Das Herzstück von GEF ist wohl der Controler der das Model mit der View verbindet. Zur Erzeugung dieser ist es nötig eine Factory zu implementieren die zu den Modelobjekten die entsprechenden Controlerobjekte erzeugt. Diese wird duch das Framework aufgerufen.

2.6.1. EditPart

Der Controler in GEF ist eine Subklasse von `EditPart`. Auch hier bildete ich eine eigene gemeinsame Superklasse welche den `GraphicalEditPart` erweitert und das Interface `PropertyChangeListener` implementiert. Dadurch konnte sich der davon spezialisierte `EditPart` immer über das Interface `PropertyChangeLister` bei dem entsprechenden Modelobjekt anmelden/bekanntmachen. Dieses Interface ist die einzige Verbindung vom Model zum Controler! Umgekehrt sah ich keine Veranlassung eine Generalisierung des Modelobjektes bei dem Controler bekannt zu machen. Der Contoler muss das Modelobjekt sehr genau kennen da er dieses bei Interaktionen (möglicherweise) verändert.

In den meisten Fällen konnte ich nun von der gemeinsamen abstrkten Superklasse `PropertyAwarwPart` ableiten. Da es den Ramen sprängen würde hier alle Methoden zu erklähre, möchte ich auf die Javadoc verweisen und nur einiger der zentralen Methoden ansprechen:

`public List getModelChildren():`

hier werden nun die Kinder des entsprechenden Modelobjektes gesucht und in einer List zurückgegeben. Diese Methode wird auch durch das Framework aufgerufen. Dies ermöglicht eine **rekursiven** Aufbau der Controlerobjekte aus den Modelobjekten. Jedes Modelobjekt welches beim Aufruf dieser Methode zurückgegeben wird, wird als Input der `EditPartFactory` übergeben und dort das entsprechende Controler- (`EdiPart`-) Objekt erzeugt. Hier bietet sich die Möglichkeit, das Mapping von Model-Contoler (-View) zu bestimmen. Beispielsweise sollen nicht alle Modelobjekte editiert (und/oder angezeigt) werden, kann dies durch die Rückgabe bestimmt werden. Der häufigste und auch einfachste Fall ist sicherlich jedoch das 1:1:1 von Model : Controler : View (jedoch keineswegs ein Muss!).
`GetModelChildren()` überschreibt hier `getModelChildren()` von `AbstractEditPart` in welcher `Collections.EMPTY_LIST` zurückgegeben wird. Ist man bei den Blättern angekommen, (im unten stehenden Beispiel bei den DBTableColumns) ist in dem entsprechenden EditPart diese Methode nicht zu überschreiben.

Beispiel: Model `DBTable` -> Controler `DBTablePart`:

```
public class DBTablePart extends PropertyAwarePart
{
        ...
        protected List<DBTableColumn> getModelChildren()
        {
                Collection<DBTableColumn> values = getTable().getColumns().values();
                return new ArrayList<DBTableColumn>(values);
        }

        ...
}
```

Über diese Liste mit DBTableColumn Objekten wird nun iteriert und einzeln der `EditPartFactory` übergeben und entsprechend die `DBTableColumnPart`-Objekte erzeugt. Diese geben dann in der von `AbstractEditPart` geerbten `getModelChildren()` eine `Collections.EMPTY_LIST` zurück.

`protected IFigure createFigure():`
Hier ist eine Subklasse von IFigure zurückzugeben. IFigure ist ein Interface Von Draw2D. Alle graphischen Repräsentationen des Models (View-elemente) müssen dieses Interface implementieren.
Beispiel DBTablePart:

```
/**
 * Creates a figure which represents the table.
 * @return IFigure
 *      The figure which represents the table.
 */
protected IFigure createFigure()
{
        DBTableFigure tableFigure = new DBTableFigure(getTable().getName());
        return tableFigure;
}
```

Später gehe ich noch genauer auf IFigure ein.

`protected void createEditPolicies():`
EditPolicy Objekte kapseln die Verarbeitung von Editieraufgaben. Dies hat zwei Vorteile:

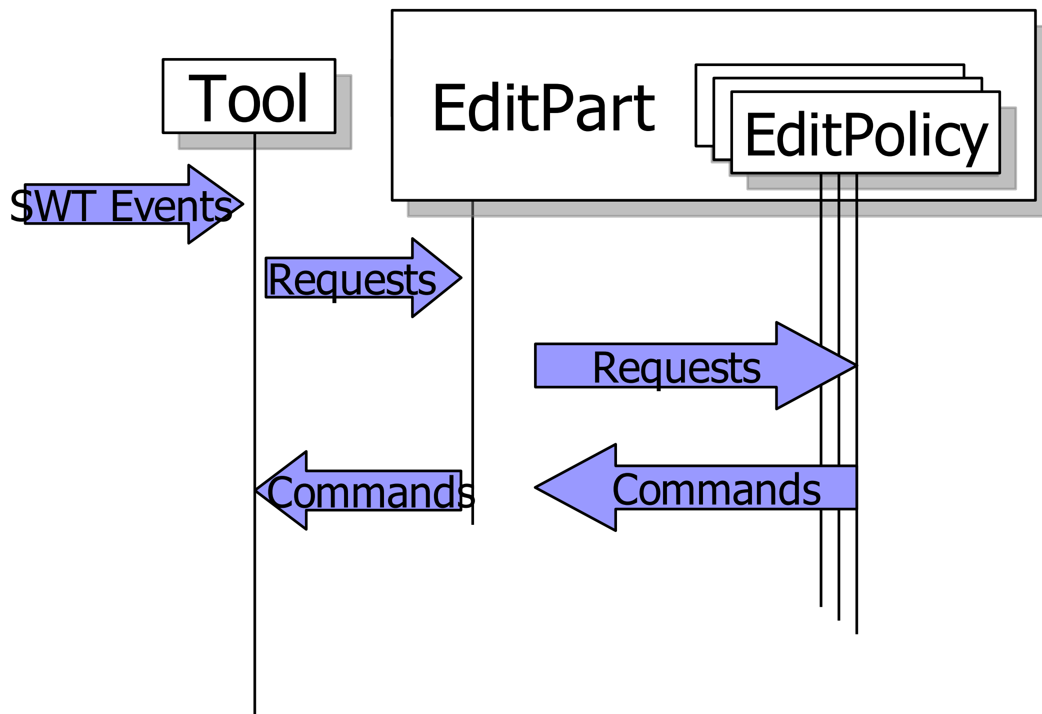
1. einer Wiederverwertung von Code. Beispielsweise ist das Editieren einer `DBTableColumn` und einer `DBViewColumn` im Wesentlichen gleich. So kann dies in einer `EditPolicy` gekapselt werden und sowohl in einem `DBTablePart` als auch in einem `DBViewPart` verwendet werden.

2. da dem Controler eine große Menge an Aufgaben zu kommt, verliert er schnell an Übersichtlichkeit. Durch den Einsatz dieser EditPolicies wird dies verbessert.

Beispiel DBTablePart:

```
/**
 * Creates edit policies and associates these with roles
 */
protected void createEditPolicies()
{
        installEditPolicy(EditPolicy.GRAPHICAL_NODE_ROLE, new ColumnNodeE-
ditPolicy());
        installEditPolicy(EditPolicy.LAYOUT_ROLE, new
TableLayoutEditPolicy());
        installEditPolicy(EditPolicy.CONTAINER_ROLE, new TableContainerEdit-
Policy());
        installEditPolicy(EditPolicy.COMPONENT_ROLE, new TableEditPolicy());
}
```

Erhält der EditPart einen Request, iteriert er über seine EditPolicies und gibt diesen weiter. EditPolicies die mit diesem Request nichts angangen ignorieren diesen. So kann ein Request auch von mehreren EditPolicies angenommen werden. EditPolicies wandeln den **Request** dann in einen **Command** um und geben diesen zur verarbeitung zurück.



Mit Hilfe der Commands ist das Redo und Undo implementiert. Sind dies Commands einmal hergestellt und an das Framework zurückgegeben, ist Redo und Undo als Funktionalität vorhanden. `execute()`, `redo()` sowie `undo()` werden in den meisten Fällen durch das Framework aufgerufen.

Commands

Eine Abstrakte Basisimplementation bietet GEF (org.eclipse.gef.comman-ds). Von dieser ist nun eine jeweilige Spezialisierung abzuleiten und die Veränderung an dem Model speichert.

2.7. Die View

2.7.1. IFigure

IFigure ist das Basisinterface von Draw2D. Alle Viewelemente implementieren dieses Interface. GEF bietet hier aine Basisimplementierung `Figure` an.

`public List getChildren()`
Wie auch bei den EditParts bieten die Figures die Möglichkeit die Kindobjekte zu erfragen.

`void add(IFigure figure)`
Um die Kinder bekannt zu machen.

`public final Dimension getPreferredSize()`
Mit Hilfe des LayoutManagers kann hier die bevorzugte Größe erfragt werden (Auchtung: die Dimension ist by value! Und sollte nicht verändert werden).

Beispiel `DBTableFigure:`

```
/**
 * The constructor.
 * @param name
 *      The name of the table.
 */
public DBTableFigure(String name)
{

        LName = new Label(name);
        ToolbarLayout layout = new ToolbarLayout();
        layout.setVertical(true);
        layout.setStretchMinorAxis(true);
        setLayoutManager(layout);
        setBorder(new LineBorder(ColorConstants.black, 1));
        setBackgroundColor(TableColor);
        setForegroundColor(ColorConstants.black);
        setOpaque(true);

        ContentPane=new ColumnsFigure();

        LName.setForegroundColor(ColorConstants.black);
        add(LName);
        add(ContentPane);
}
```

Die ContentPane ist hier der Platz in welchen die Columns bei `add(IFigure figure)` eingefügt werden.

2.8. Anschluß der Outline und der PropertyView

Der Anschluß der Outline war über die Möglichkeit die Outline als `SelectionChangeListener` bei dem `GraphicalViewer` (eine art Verwaltungsobject welches über den Zustand der EditParts bescheid weiß) des Editors (EditDomain) zu registrieren und umgekehrt, ohne große Probleme möglich. So kann über die Outline navigiert werden und das entsprechende Viewobjekt und die entsprechende PropertyPane angezeift werden.

Den Anschluß der EditPanes war aus zeitlichen Gründen nicht mehr möglich. Hierzu könnte man die Validators, die bei der Eingabe von Daten verwendet werden, durch EditPolicies ersetzen und den EditPart das Interface des Validators implementieren lassen. So sollten die Events die von den PropertyParts ausgehen von dem entsrechenden EditPart object abgefangen werden und an den zuständigen EditPolicy weitergeleitet werden. Durch die Ezeugung eines entsprechenden Commans zur veränderung des Modelobjektes wäre auch hier die Redo/Undo-Funktion gegeben.

2.9. Lessens learned

Was mir zu Beginn etwas Halt gab, wurde mir zum Ende zur Last: ich trennte mich nicht früh genug von dem Beispielplugin um einen sauberen Code zu erzielen. So habe ich nun noch einige Fragmente von undurchforstetem Code übrig die ich noch nicht beseitigt habe, da ich ihre genaue Funktion noch nicht erkundet habe.

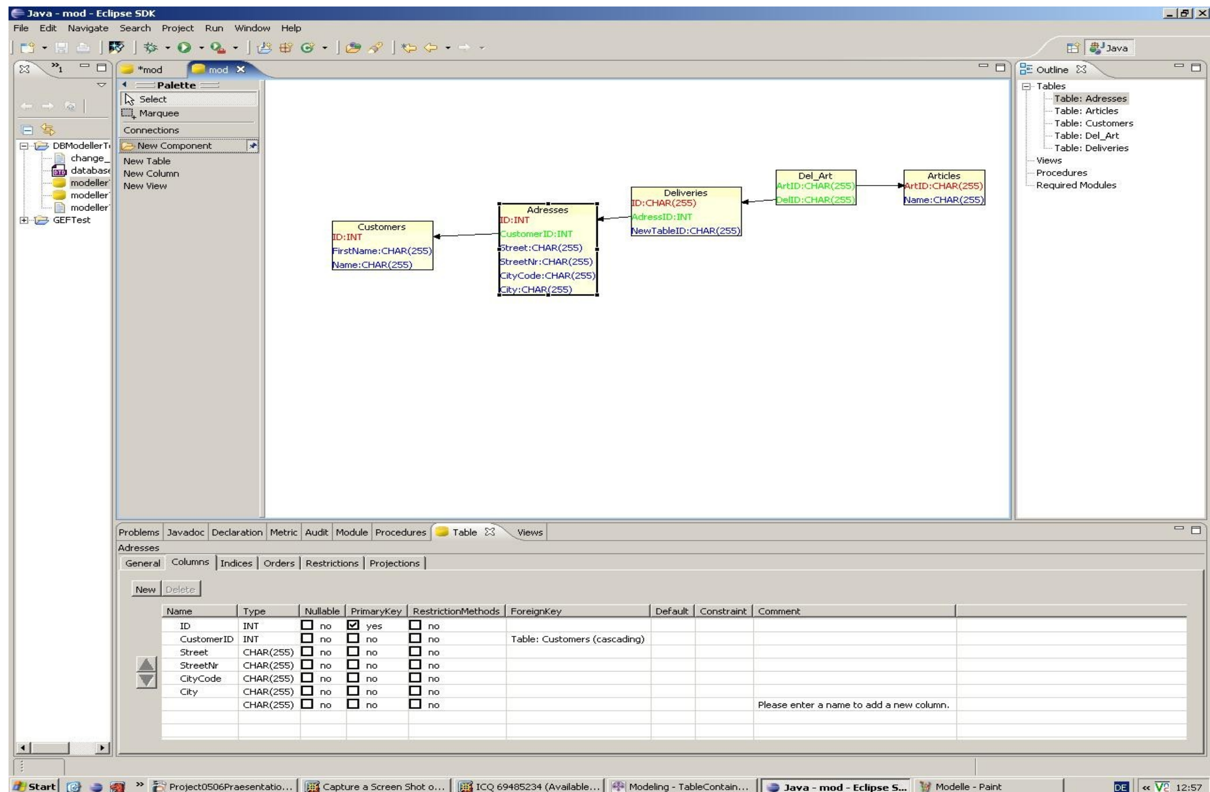
1. Für ein weiteres Projekt dieser Art würde ich versuchen mich von dem Beispiel rechtzeitig zu trennen und nur die sicher benötigten Codepartien einzubeziehen.
2. Da ich dieses Projekt alleine realisiert habe, war ich sehr flexibel, musste mich mit niemandem absprechen, allerdings verteilte sich die Arbeit auch ausschließlich auf mich selbst ;-(. Die Erfahrungen beim Arbeiten in bzw. mit einem Team blieben daher völlig aus, was ich als nachteilig sehe und bei einem weiteren Projekt hier an der Fachhochschule (und auch ausserhalb) anders handhaben würde.
3. Dokumentierter Code ist super! Vor allem wenn der Text auch aussagekräftig ist. (Teilweise fiel es mir schwer mich in dem Wald von Klassen, Interfaces und Methoden zurechtzufinden, da es nicht nur viele gab sondern teilweise auch die Kommentare nicht wirklich hilfreich waren).
4. Eine zähe und schwierige Einarbeitung in ein Framework rentiert sich und bringt Spaß.

5. DesignPatterns gibt es nicht bei den GoF sondern auch angewad in z.B. diesem Framework. Durch das Kennen einiger dieser Patterns fiel mir das Verständnis für den Aufbau des Frameworks deutlich leichter.
6. Strukturieren macht das Leben leichter.
7. **Steile und hohe Lernkurfe, dann aber viel Spaß und viele Möglichkeiten!**

2.10. Fazit

Das Graphical Editing Famework ist ein mächtiges Werkzeug für die interaktive Visualisierung von Sofwaremodellen, das mit beliebigen Objekten abeiten kann, da es keine Annahmen über die Modelstruktur trifft.
Die Einarbeitung ist durch die etwas düftige Dokumentation für das relativ große Framework recht mühsam. Durch die klare Architektur, besonders durch die konsequente Einhaltung des MVC-Paradigmas und die modulare Integration der verschiedenen Features wird die Einarbeitung jedoch erleichtert.
Die Mächtigkeit von GEF schlägt sich auch in der Komplexität des Frameworks nieder. Der recht hohe Einarbeitungsaufwand wird jedoch belohnt, da es sehr flexibel einsetzbar ist und **häufig wiederkehrende Probleme elegant lößt**.
Eine Erweiterung des Frameworks auf spezielle Bedürfnisse ist vermutlich durch den modularen Aufbau ohne große Probleme möglich.

In meinem Fall trug die Portierung auch zu einer besseren Strukturierung und dadurch zu einer besseren Wartbarkeit des Code bei. Die anfangs erwähnten, oft recht aufwendigen Zeichenroutinen und Eventinterpretationen bei den Interaktionen werden einem fast geschenkt.
Aus Zeit und Platzgründen habe ich hier nur sehr elementare Dinge beschrieben. Schon alleine durch das Beschäftigen mit dem Framework über das Semester hinweg, wäre es sicherlich möglich ein halbes Buch zu schreiben und es wäre dann noch bei weitem noch nicht alles über GEF (und Draw2D) gesagt.
Wer den etwas mühsamen Einstig nicht scheut wird sicherlich viel Spaß damit haben und viele Einsatzmöglichkeiten erkennen.



3. Ausblick

- Ich selbst sehe mich auch nach der intensiven Beschäftigung über das Semester hinweg noch in der Einarbeitungsphase. Durch die klare Architektur und die vielen Einsatzmöglichkeiten würde ich diese gerne noch weiter treiben und an Durchblick dazugewinnen.

- Diesen Prototypen mal noch zu einem produktiven Tool verfollständigen.

- Neugierig bin ich auch auf das neu entwickelte Graphical Modeling Framework in Eclipse welches das EMF (Eclipse Modeling Framework) und das GEF zusammenbringt.

Die Zukunft bleibt spannend!

4. Quellen

- Eclipse: http://www.eclipse.org/gef das „Zuhause“ von GEF ...
  /* hier findet man alles rund um GEF */

- IBM Redbook http://www-106.ibm.com/developerworks/opensource/library/os-gef/ „Create an Eclipse-based application using the Graphical Editing Framework“ von Randy Hudson
  /* Ein guter Einstieg in GEF – Grundlagen */

- Building a Database Schema Diagram Editor with GEF
  /* Dieser Artikel und das dazugehörige Beispielplugin diente mir als Grundlage */

- Eclipse Magazin (http://www.eclipsemagazin.de/) Vol.2: „Maßgeschneiderte graphische Editoren mit dem Graphical Editing Framework“
  /*Sehr guter Artikel/Tutorial von Dr. Boris Bokowski und Dr. Frank Gerhardt */

- Vortrag im Rahmen der Java User Group Stuttgart (http://www.jugs.org/), gefunden http://www.eclipseteam.de/wiki/bin/view/Public/EclipseGef von Dr. Boris Bokowski

- „Contributing to Eclipse“, Erich Gamma und Kent Beck
  /* Beschäftigt sich nicht mit GEF aber der Pluginentwicklung für Eclipse */

- EclipseCon http://www.eclipsecon.org/2005/tutorials.php „GEF in Depth“
  /* Einige Folien als mögliche Ergänzung, jedoch ohne Worte eher etwas mühsam zum nachvollziehen */

5. Der DB-Modeller

Diese kurze Beschreibung das DB-Modeller als Eclipse Plug-in soll Ihnen einen groben Überblick über die enthaltenen Features bieten und den Einstieg erleichtern. Bei der Entrwicklung wurde großen Wert auf eine intuitive Bedienbarkeit des Plug-ins geachtet, so dass ich hoffe mit dieser Übersicht die möglichen Fragen zu klären.

5.1. Überblick

Der DB-Modeller bietet die Möglichkeit der graphischen Datenbankdarstellung und -modellierung unter Verwendung des Entity-Relationship-Modell (ERM). Dies soll dem Benutzer einen visuellen Überblick über die verschiedenen Entities, deren Beziehungen untereinander (Foreign-Key) ermöglichen und Modellierungsmöglichkeiten bieten.
Die Besonderheit des DB-Modeller ergeben sich aus den Datenbankdefinitionsmöglichkeiten (DatabaseModule.dtd) der Ebdac Software GmbH. Hier besteht die Möglichkeit, schon im DB-Design zusätzlich zu Tables und Views noch Procedures, Restrictions Projections, Orders und RequiredModules sowie Comments anzugeben. Dadurch ist es möglich, entsprechende Java-Klassen durch eine automatische Codegenerierung zu erzeugen.
Sollten irgendwelche Änderungen (z.B. eines SQL-Statements, ...) nötig sein, genügt es diese im DB-Modeller bzw. im entsprechenden XML-File anzugeben. Die betroffene(n) Java-Klasse(n) wird(werden) dann entsprechend erzeugt. Eine geringere Fehleranfälligkeit ist dadurch möglich.

Ziel des DB-Modeller in Version 1.0.0 ist es, die spezifischen Anforderungen hinsichtlich der bestehenden Datenbankdefinitionsmöglichkeiten der Ebdac Software GmbH durch das Ebdac Databes Framework in einer GUI abzubilden, sowie eine funktionelle und benutzerfreundliche Oberfläche zu bieten.

5.2. DB-Modeller als Eclipse Plug-in

Als technische Grundlage wird die als Open–Source verfügbare IDE Eclipse* (Vers. 2.1.2) verwendet. Eclipse als Plattform bzw. Framework bietet hierfür die nötigen Ressourcen, auf welche über entsprechende Erweiterungspunkte zugegriffen werden kann. Ein Plug-in wie hier nützt diese Ressourcen und erweitert diese um die zusätzlich gewünschte Funktionalität.

5.3. Die DatabeseModule.dtd

Aufbau der (Ebdac) DB-Definitionsmöglichkeiten:

```
<!ELEMENT DatabaseModule (Tables?,Views?,Procedures?,Restric-
tions?,Projections?,Orders?,RequiredModules?,Comment?)>
```

In dem oben gezeigten Ausschnitt der DatabaseModule.DTD wird ein Datenbank-Modul durch die Angabe von keinem, einem oder mehreren Tables, Views, Procedures, RestrictionsProjections, Orders, RequiredModules und Comment vollständig beschrieben.
Im Folgenden möchte ich einige der mir wichtig erscheinenden Definitionen kurz vorstellen und erläutern. Die vollständige DatabaseModule.dtd ist im Anschluss zu finden.

5.3.1. Tables

```
<!ELEMENT Table (Columns,Indices?,Comment?)>
```

5.3.2. Views

```
<!ELEMENT Views (View*)>
      <!ELEMENT View (SQLStatement,ResultColumns,Comment?)>
→                  <!ELEMENT SQLStatement (#PCDATA)>
```

Hier im Element SQLStatement liegt eine mögliche Fehlerquelle, da die Statements beim Design noch nicht auf ihre Gültigkeit hin überprüft werden können. Dies ist wohl die häufigste Fehlerquelle, die eben erst nach der Erzeugung der Datenbank überprüft werden kann.

5.3.3. Procedures

```
<!ELEMENT Procedures (Procedure*)>
    <!ELEMENT Procedure (Parameters?,SQLStatement,ResultColumns?,Com-
ment?)>
```

Procedures hier sind im Unterschied zu Views nicht persistent in der Datenbank. Sie bieten zusätzlich die Möglichkeit der Parameterangabe. Sie dienen nicht nur für SELECT-Anweisungen sondern auch für DELETE und UPDATE (-> alle Anweisungen der Data-Modelling-Language)

5.3.4. Restrictions

```
<!ELEMENT Restrictions (Restriction*)>
    <!ELEMENT Restriction (Parameters?,WhereClause,Comment?)>
```

Bietet Restriction die Möglichkeit der Zeilenauswahlwahl, so kann mit Projections eine Auswahl an Spalten von einer Tabelle bestimmt werden.

5.3.5. Projections

```
<!ELEMENT Projections (Projection*)>
    <!ELEMENT Projection (ColumnNames,Comment?)>
```

5.3.6. Die vollständige DatabaseModule.dtd

```
<!ELEMENT DatabaseModule (Tables?,Views?,Procedures?,Restric-
tions?,Projections?,Orders?,RequiredModules?,Comment?)>
<!ATTLIST DatabaseModule
          Name CDATA #REQUIRED
          Version CDATA #REQUIRED
          Database CDATA #REQUIRED
          PackageName CDATA #IMPLIED>
  <!ELEMENT Tables (Table*)>

    <!ELEMENT Table (Columns,Indices?,Comment?)>
    <!ATTLIST Table
              Name CDATA #REQUIRED
              SingularName CDATA #IMPLIED
              SetClass CDATA #IMPLIED
              ObjectClass CDATA #IMPLIED
              ObjectSuperClass CDATA #IMPLIED
              ObjectInterfaces CDATA #IMPLIED>

      <!ELEMENT Columns (Column+)>

        <!ELEMENT Column (ForeignKey?,Comment?)>
        <!ATTLIST Column
                  Name CDATA #REQUIRED
                  Type CDATA #REQUIRED
                  Nullable (yes|no) #REQUIRED
                  Default CDATA #IMPLIED
                  Constraint CDATA #IMPLIED
                  PrimaryKey (yes|no) "no">

          <!ELEMENT ForeignKey EMPTY>
          <!ATTLIST ForeignKey
                    Table CDATA #REQUIRED
                    Column CDATA #REQUIRED
                    Delete (NoAction|Cascade) #REQUIRED>

      <!ELEMENT Indices (Index*)>

        <!ELEMENT Index (ColumnNames,Comment?)>
        <!ATTLIST Index
                  Name CDATA #REQUIRED
                  Unique (yes|no) "no"
                  Clustered (yes|no) "no">

          <!ELEMENT ColumnNames (ColumnName+)>

            <!ELEMENT ColumnName EMPTY>
            <!ATTLIST ColumnName
                      Name CDATA #REQUIRED>

      <!ELEMENT Comment (#PCDATA)>

  <!ELEMENT Views (View*)>

    <!ELEMENT View (SQLStatement,ResultColumns,Comment?)>
    <!ATTLIST View
              Name CDATA #REQUIRED
```

```
                SingularName CDATA #IMPLIED
                SetClass CDATA #IMPLIED
                ObjectClass CDATA #IMPLIED
                ObjectSuperClass CDATA #IMPLIED
                ObjectInterfaces CDATA #IMPLIED>

        <!ELEMENT SQLStatement (#PCDATA)>

        <!ELEMENT ResultColumns (ResultColumn+)>

          <!ELEMENT ResultColumn (Comment?)>
          <!ATTLIST ResultColumn
                    Name CDATA #REQUIRED
                    Type CDATA #REQUIRED
                    SourceTable CDATA #IMPLIED
                    SourceColumn CDATA #IMPLIED>

    <!ELEMENT Procedures (Procedure*)>

      <!ELEMENT Procedure (Parameters?,SQLStatement,ResultColumns?,Com-
ment?)>
      <!ATTLIST Procedure
                Name CDATA #REQUIRED
                Result (Nothing|ResultSet) #REQUIRED
                SingularName CDATA #IMPLIED
                SetClass CDATA #IMPLIED
                ObjectClass CDATA #IMPLIED
                ObjectSuperClass CDATA #IMPLIED
                ObjectInterfaces CDATA #IMPLIED>

        <!ELEMENT Parameters (Parameter*)>

          <!ELEMENT Parameter (Comment?)>
          <!ATTLIST Parameter
                    Name CDATA #REQUIRED
                    Type CDATA #REQUIRED
                    MultipleValues (yes|no) "no">

    <!ELEMENT Restrictions (Restriction*)>

      <!ELEMENT Restriction (Parameters?,WhereClause,Comment?)>
      <!ATTLIST Restriction
                Table CDATA #REQUIRED
                Name CDATA #REQUIRED>

        <!ELEMENT WhereClause (#PCDATA)>

    <!ELEMENT Projections (Projection*)>

      <!ELEMENT Projection (ColumnNames,Comment?)>
      <!ATTLIST Projection
                Table CDATA #REQUIRED
                Name CDATA #REQUIRED>

    <!ELEMENT Orders (Order*)>

      <!ELEMENT Order (OrderColumns,Comment?)>
      <!ATTLIST Order
```

```
                Table CDATA #REQUIRED
                Name CDATA #REQUIRED>

        <!ELEMENT OrderColumns (OrderColumn+)>

          <!ELEMENT OrderColumn EMPTY>
          <!ATTLIST OrderColumn
                    Name CDATA #REQUIRED Sort (Asc|Desc) "Asc">

    <!ELEMENT RequiredModules (RequiredModule*)>

      <!ELEMENT RequiredModule EMPTY>
      <!ATTLIST RequiredModule
                Name CDATA #REQUIRED
                Version CDATA #REQUIRED>
```

5.4. Die Installation des Plug-ins

Ein weiterer Vorteil von Eclipse als Plattform ist das einfache Installieren und Deinstallieren von Plug-ins. Zum installieren des Plug-ins genügt es das .zip-File in das Plug-ins Verzeichnis im Eclipse Verzeichnis zu entpacken. Wird das Plug-in in Eclipse 2.x installiert ist ein Neustart von Eclipse notwendig. Ab der Version 3.0 ist ein Neustart nicht mehr notwendig.
Soll das Plugin entfernt werden, müssen nur die entsprechenden Dateien (hier com.ebdac.db-modeller) aus dem Eclipse Pluginverszeichnis gelöscht werden.

5.5. Dataiformate

5.5.1. Das .xml-File

Zur DBMS unabhängigen Speicherung der Daten wird ein XML-File verwendet.
Beim Import einer Datei überprüft der DB-Modeller diese auf den für ihn richtigen Doctypes.

Die Speicherung der DBMS-Daten erfolgt im selben XML-File und die Daten zur graphischen Darstellung in einem separaten .dbm-File, um Datenbankdaten und Darstellung getrennt verwenden zu können.

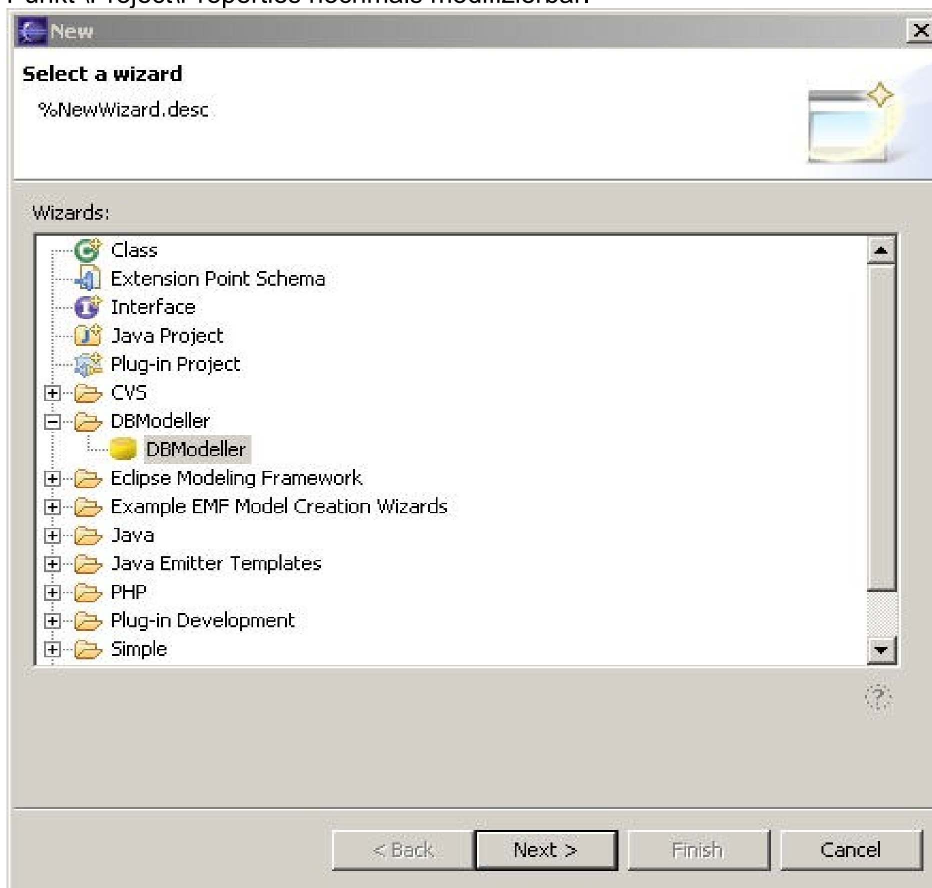
5.5.2. Das .dbm-File

Das .dbm-File mit den Informationen zur graphischen Darstellung beschreibt zusätzlich das dazugehörige .xml-File mit den Datenbankbeschreibung, so dass diese beim nächsten öffnen wieder übernehmen werden können.

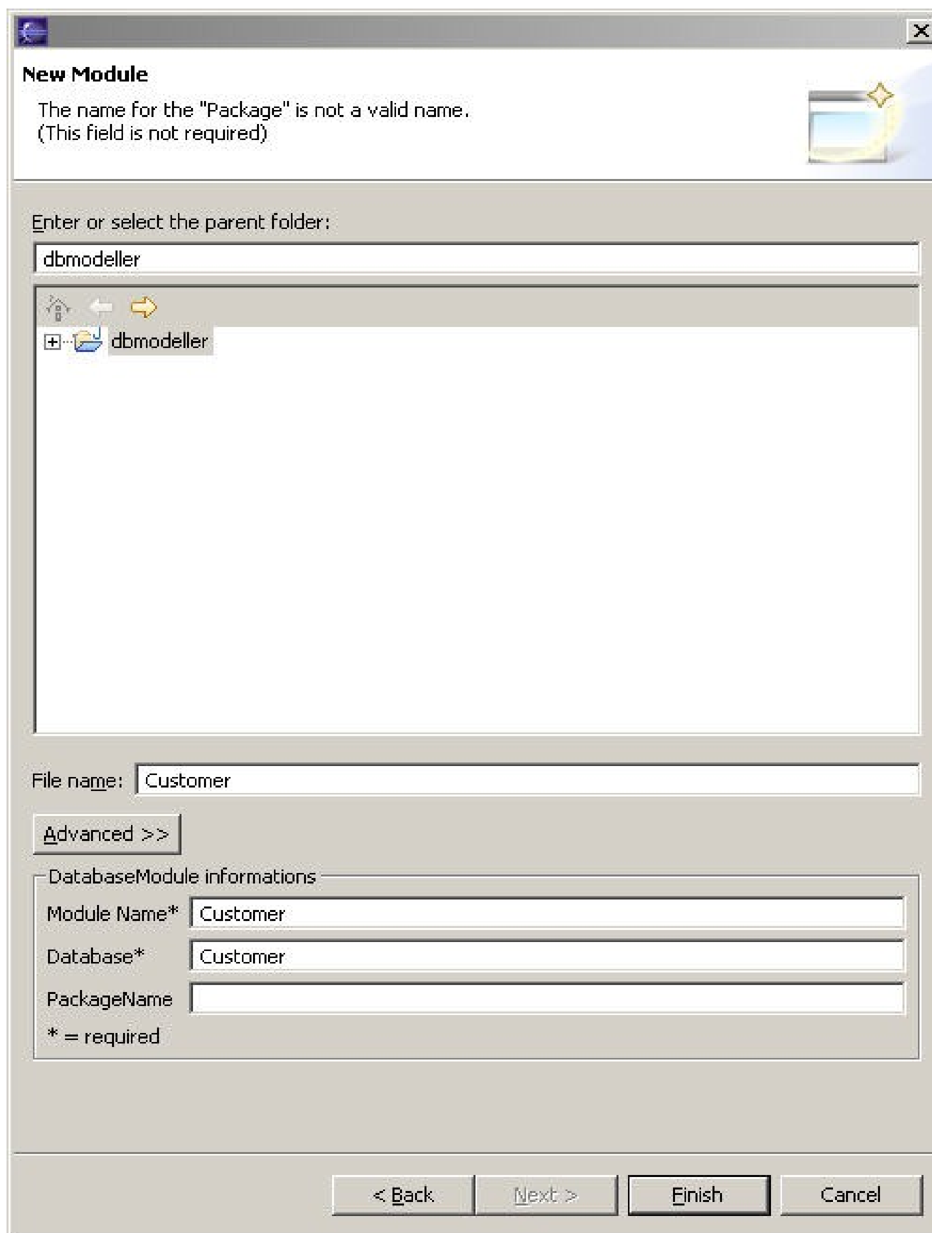
5.6 Editieren

5.6.1. Neues Modul anlegen

Zum Anlegen eines neuen Moduls wird über \File\ New\ Other\ DBModeller ein Wizard gestartet, durch den die gewünschten Einstellungen angegeben werden können. Hier können bereits vorhanden DBMS-Daten in form eines XML-Files über einen Dateibrowser ausgewählt oder ein neues XML- und dbm-File angelegt werden. Diese sind anschließend unter dem Punkt \Project\Properties nochmals modifizierbar.



File/New/Other

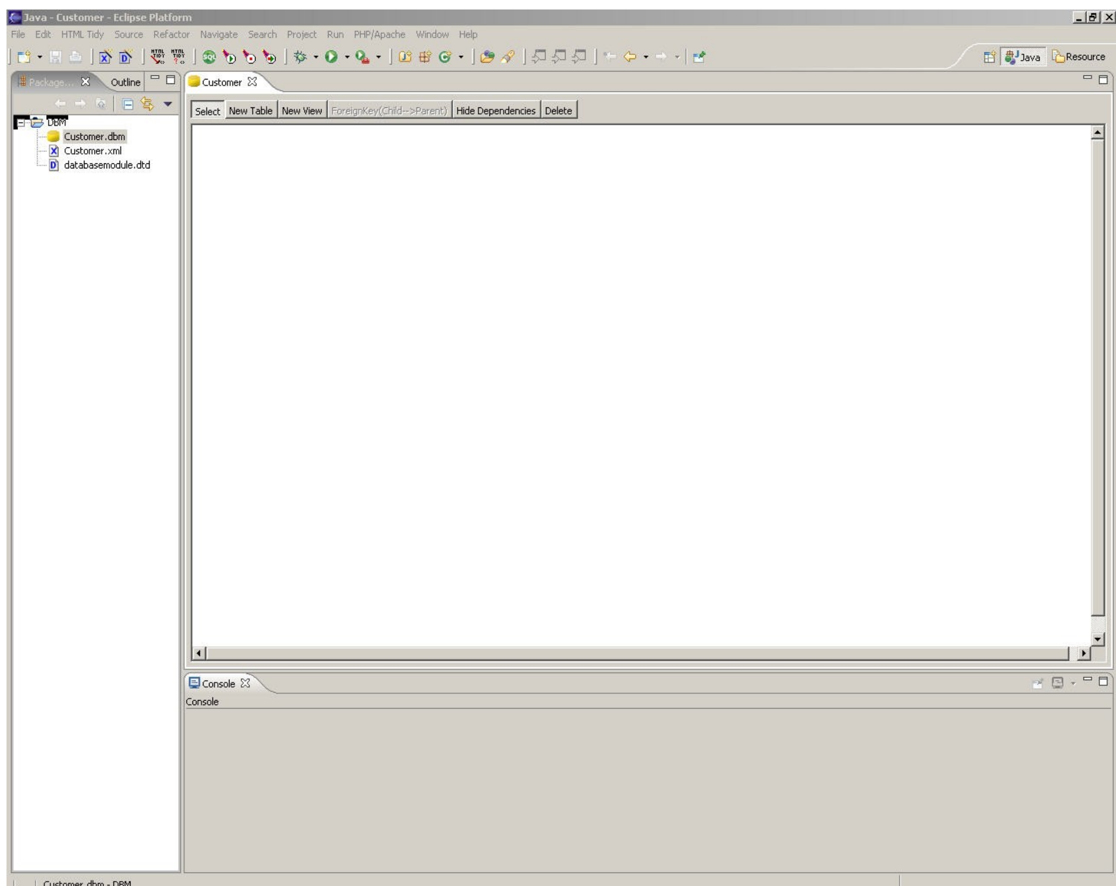


5.6.2. Modelling -View

In dieser View werden die Tables, Views und Foreign-Key-Beziehungen graphisch dargestellt und modelliert. Die Tabellen oder Views können direkt aus der Icon-Bar über Klick und Set auf dem Editor erzeugt und platziert werden. Eine erneute Platzierung ist über Drag&Drop möglich.

Zunächst wird beim doppelklicken der .dbm-Datei oben gezeigte Fehlermeldung erscheinen, da die benötigte databesemodule.dtd nicht im Projekt liegt. Diese muss noch über File\Import\Filesystem importiert oder in das Projekt im Eclipse workspace kopiert werden.

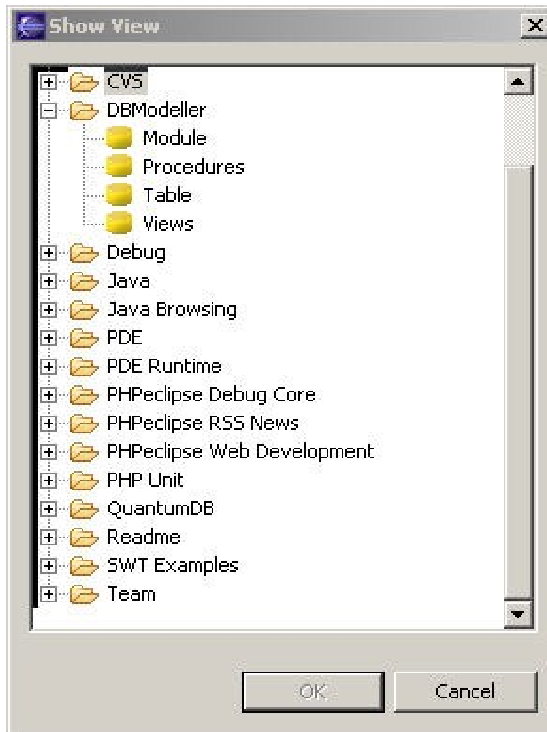
Ist diese nun bekannt, erscheint bei einem neu angelegten .dbm-File ein leerer Editor, auf dem nun Tables und Views angelegt und bearbeitet werden können.
Dies geschieht über das Aktivieren des NewTable- bzw. NewView Buttons in der Werkzeugleiste und anschließendem Platzieren im Editor.



Die jeweiligen Eigenschaften der Elemente werden im Edit-Pane angezeigt. Optionen des aktiven Elementes kann über Drop-Down-Menüs bzw. Buttons ausgewählt werden.

5.7. Edit-Views

5.7.1. Anzeigen der Edit-Views



Zunächst sollten über Window/Show View/Other/DBModeller die gewünschten Views (Module, Procedure, Table, Views) hinzugefügt werden.

5.7.2. Model View

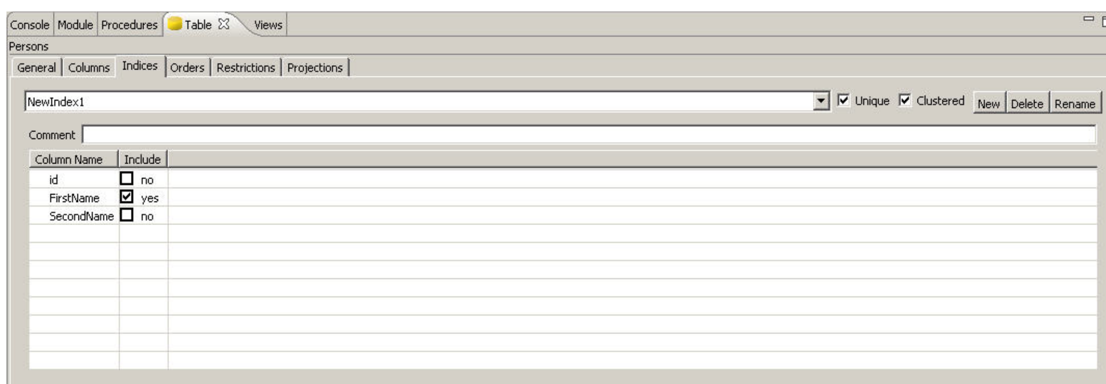
In der Model View können nun die Grundsätzlichen Eigenschaften angegeben werden. Hier sind die im Wizard zu Beginn angegebenen Informationen hinterlegt. Diese können hier nachträglich geändert werden.

5.7.3. Table View

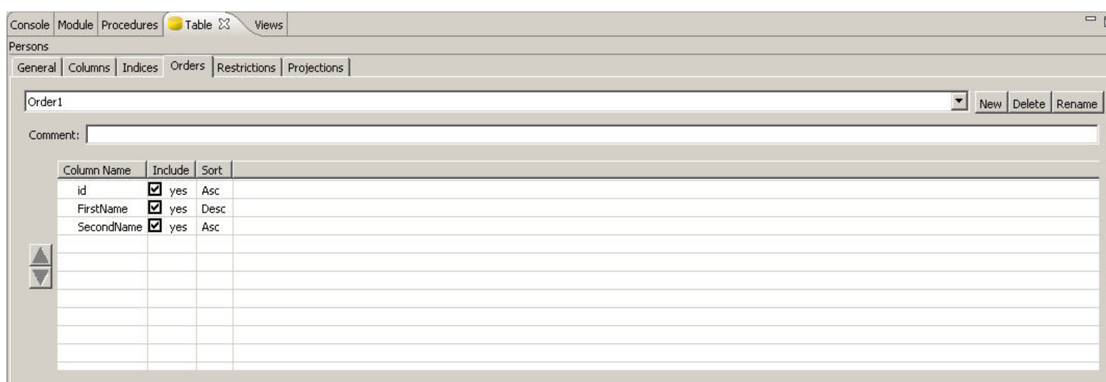
Die View zum Bearbeiten der in einem Table enthaltenen Elemente und derer Eigenschaften ist in den verschiedenen Tabs innerhalb der Table View möglich.

Unter General können die den ganzen Table betreffenden Angaben gemacht werden; insbesondere der Name des Tables über den Button und einer Dialogbox geändert werden.

Über Columns können können dem momentan aktiven Table Columns hinzugefügt oder entfernt werden, sowie die Eigenschaften dieser angegeben weden. Dazu werden entweder die gewünschten Checkboxen aktiviert, oder aus dem Dropdown Menü welches sich beim aktivieren der entsprechenden Zelle auftut, der gewünschte Wert ausgewählt.



In der Indices View werden die gewünschten Indices angelegt und die beteiligten Columns über Checkboxen ausgewählt.



Orders können über den gleichnahmigen Tab angegeben, die beteiligten Columns ausgewählt und die Sortierrichtung (Ascending – Descending) angegeben werden.

5.8. Tables und Views

Die Tabellen oder Views können direkt aus der Icon-Bar über Klick und Set auf dem Editor erzeugt und platziert werden. Eine erneute Platzierung ist über Drag&Drop möglich. Eine unterschiedliche Anzeigeform von Tables und View kann unter \Project\Properties angegeben werden.

5.8.1. Primary-Key

Der Primary-Key einer Tabelle wird im Editor als Column angelegt und durch die Aktivierung des Primary-Key-Feldes in der Table View als solche definiert. Graphisch wird diese Column nun besonders angezeigt. Jede Table kann max. einen PK definieren. Beim Löschen eines

Primary-Keys werden eventuell vorhandene Foreign-Keys aus den referenzierenden Tables nach einer Warnmeldung gelöscht. Dies ist abhängig von der aktivierung des cascading delete bei der Angabe der Foreign-Keys (s. 2.2.4).

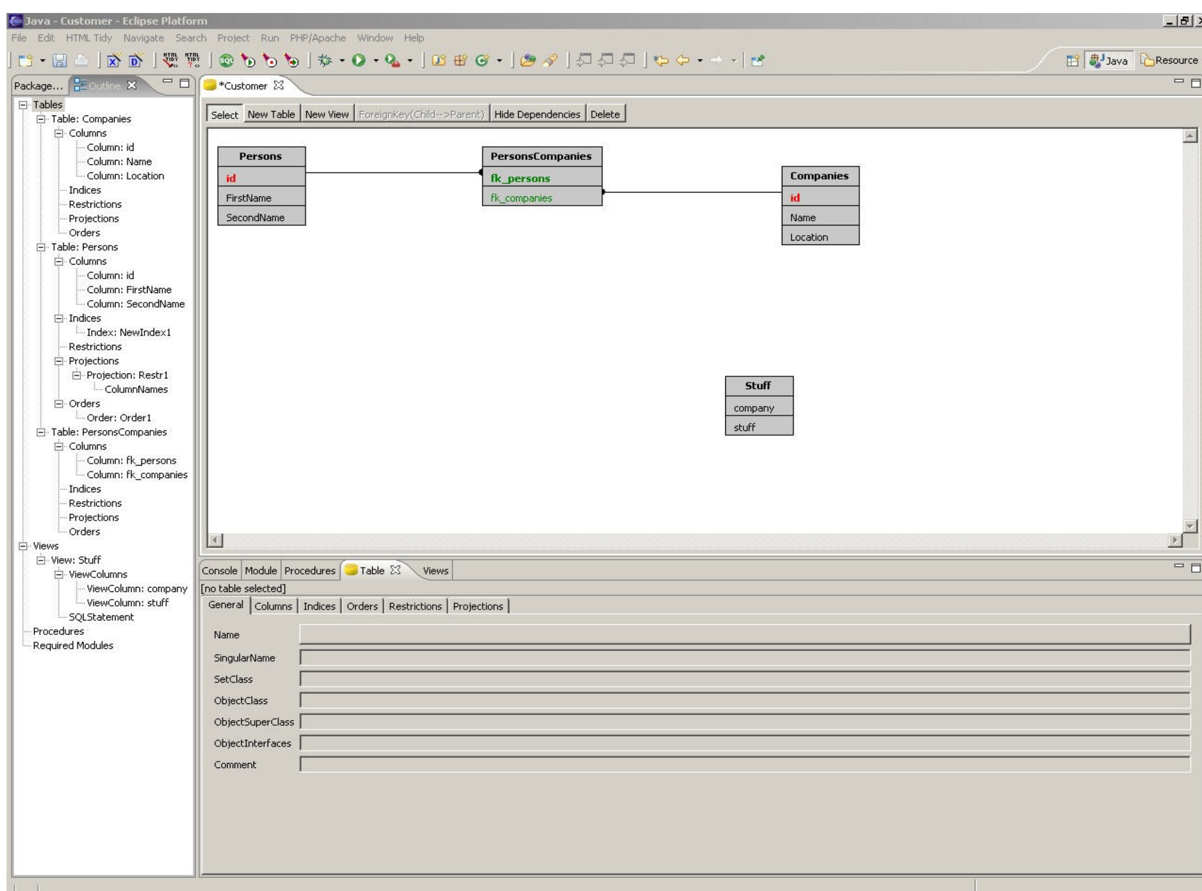
4.8.2. Foreign-Key

ForeignKeys können per Klick und Klick der jeweiligen Tabellen erzeugt werden. Diese sind wie auch die Primary-Keys durch eine andere typographische Darstellung hervorgehoben, was einer schnelleren Übersicht dient.
Hierbei definiert der erste Klick die referenzierende, der zweite Klick die referenzierte Tabelle. Visualisiert wird die Foreign Key-Beziehung durch eine Linie von der Foreign-Key-Spalte ausgehend, hin zur Primary-Key-Spalte der referenzierten Tabelle. Sollte dieser noch nicht angegeben worden sein, erscheint der Dialog zur Editierung dieser Tabelle. Wurde die Foreign-Key-Spalte in der referenzierenden Tabelle noch nicht angelegt, wird diese automatisch hinzugefügt.
Wird der Primary-Key aus der referenzierten Table gelöscht, wird auch der/ die entsprechende Foreign-Key entfernt. Diese Änderungen können sowohl aus der Edit-Pane, als auch aus der Table-View heraus gemacht werden. Eine konsistente Darstellung ist durch das zentrale XML-File gewährleistet.
In der Table View kann durch das aktivieren der Foreign Key/Column Zelle aus einer Dropdownliste mit den Tables die einen Primary Key enthalten, der zu referenzierende ausgesucht werden. Anschliessend ist noch anzugeben, ob die Foreign Key Column auch gelöscht werden soll für den Fall dass die entsprechende Primary Key Coulumn gelöscht wird (cascading delete).

Ggf. wird der Typ der Column mit dem Foreign Key auf den Typ der entsprechenden Primary Key Column umgestellt (referenzierende und referenzierte Spalten müssen vom gleichen Typ sein).

4.9. Outline

Das Outline Fenster stellt die im XML-File gespeicherten Elemente dar. Hierüber ist ein schnelles navigieren möglich was besonders für in Tables und Vies enthaltene Element sowie für Procedures interessant ist. Beim anklicken eines Elements in der Outline wird die entspre-chende View und das entsprechnde Tab in dieser aktiviert.



Wird ein Element in einer Ansichten (Editor, View oder Outline) aktiviert wird das entspre-chende Element in den anderen Ansichten ebenfalls aktiviert. (in der obigen Abbildung ist kein Element aktiviert!).

6. Anlagen:

Quellcode
Javadoc
PDF der Präsentation am 28.01.2006

auf der beiliegenden CD