HTBLuVA Wiener Neustadt
Höhere Lehranstalt für Informatik

DIPLOMARBEIT

Entity Relationship Modelling Toolkit ERD

Ausgeführt im Schuljahr 2018/19 von:

Berndt FISCHBACHER	5BHIF
Christian PASSET	5BHIF
Andreas PRINZ	5BHIF
Nicolas HOMOLKA	5BHIF

Betreuer / Betreuerin:

Dipl.-Ing. Günther Burgstaller

Wiener Neustadt, am 26. März 2019

Abgabevermerk: Übernommen von:

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Wiener Neustadt, am 26. März 2019

Verfasser / Verfasserinnen:

Berndt FISCHBACHER

Christian PASSET

Andreas PRINZ

Nicolas HOMOLKA

Inhaltsverzeichnis

Vorwort

0.1 Zielsetzung

Author:

Teil I

Einführung

Kapitel 1

Aufgabenstellung

→*Passet*

1.1 Auslöser

Das ERD (Entity-Relationship-Diagramm) ist seit seiner Entstehung nicht mehr vom Prozess des Datenbank-Designs wegzudenken. Durch dieses Diagramm lässt sich leichter ein konzeptioneller Entwurf für eine Datenbank entwickeln.

Das Erstellen eines solchen Diagramms ist, bis zu einer gewissen Größe der Datenbank noch per Hand bewältigbar, wird jedoch mit ansteigenden Anzahl von Entitäten schnell komplex und zudem auch mühsam, im Bezug auf die Anordnung der einzelnen Elemente innerhalb des Diagramms.

Aus diesem Grund hat Prof. DI Günter Burgstaller das Diplomarbeitsteam beauftragt ein englischsprachiges Command-Line-Tool zu entwickeln, welches die Erstellung eines solchen Entity-Relationship-Diagramms vereinfachen soll.

1.2 Einsatz und Nutzen

Dadurch, dass der Auftraggeber Prof. DI Günter Burgstaller das Fach Datenbanken und Informationssysteme an der HTBLuVA Wiener Neustadt unterrichtet, kann das Tool in seinem Unterricht durchaus zum Einsatz kommen.

Der Nutzen den man sich von diesem Tool verspricht ist es, sich ein Diagramm in Form von verschiedenen Ausgabeformaten automatisch zeichnen zu lassen, was einem Zeit ersparen soll. Für den Unterricht kann dies bedeuten, dass die Schüler in kürzerer Zeit lernen ein ERD zu lesen und es richtig zu modellieren.

Des Weiteren wird durch die Bereitstellung des zu zeichnenden Modells in XERML, einem von Prof. DI Günter Burgstaller entwickeltem XML-Vokabulars, eventuell der Zugang zu XML für die Schüler erleichtert, was auch eine Entlastung für den unterrichtenden Professor sein kann.

Kapitel 2

Wieso das ER-Modell?

→Passet

2.1 Allgemeines zu semantischen Datenmodellen

Ein Großteil der semantischen Datenmodelle wurde in den 1970er entwickelt. Sie dienen der Erstellung einer ersten formalen Beschreibung der Datenbank und werden im Buch „**taschenbuch**" wie folgt definiert:

> Semantische Datenmodelle beschreiben einen Weltausschnitt als Menge von Gegenständen (Objekten), zwischen denen wohldefinierte Beziehungen existieren und die durch Eigenschaften charakterisiert werden.

Durch die Veranschaulichung von Informationen gehören die semantischen Datenmodelle in der Informationsmodellierung zu den Standards. Das am weitest verbreitete ist das Entity-Relationship-Modell von Peter P. Chen.

2.2 Entity-Relationship-Modell

2.2.1 Grundkonzept

Im Grunde besteht das ER-Modell aus einer handvoll an Elementen:

- Das zu modellierende Objekt genannt Entity und dessen Typ
- Die Beziehung oder Relationship und deren Typ
- Attribute welche die Eigenschaften eines Entitytypen repräsentieren

In der nachfolgenden Abbildung wird die minimalistischste Version eines ER-Diagramms gezeigt, welches die grafische Darstellung des ER-Modells ist. Das Beispiel selbst gibt den Sachverhalt wieder, dass die Entitytypen „Baum" und „Blatt" über den Beziehungstyp „hängt" miteinander assoziieren.

Abbildung 2.1: Simples Beispiel für ein ER-Diagramm

Instanzebene

→*Passet*

Bei einem Entity handelt es sich stets um ein eindeutiges, abgrenzbares Objekt aus der realen Welt oder anders formuliert, um die zu repräsentierende Informationseinheit innerhalb einer Datenbank. In einem ER-Diagramm werden jedoch nicht die einzelnen Entitäten dargestellt, sondern eine Menge von ihnen, die über den Entitytyp definiert sind.

Zwischen den Entitäten sind Beziehungen definiert, die ebenfalls nicht einzeln, sondern durch den dazugehörigen Beziehungstyp in einem Diagramm angezeigt werden.

Eine nähere Beschreibung von Entitäten und Beziehungen erfolgt durch die Verwendung von Werten. Jene Werte werden wiederum in Werte-mengen zusammengefasst und durch Wertebereiche definiert.
Ein Wertebereich wird im Normalfall durch einen Standard-Datentyp beschrieben. Die Gängigsten darunter sind:

- Char für die Darstellung von Zeichenketten
- Integer für ganze Zahlen
- Date für Datumswerte

In der Abbildung 2.2 werden, mit einem Beispiel aus dem Buch **taschenbuch**, die Entitäten vom Typ „Lieferant" durch die Werte „Koch & Krug" und „H. Schulze" und die Beziehung durch den Wert „liefert" dargestellt.

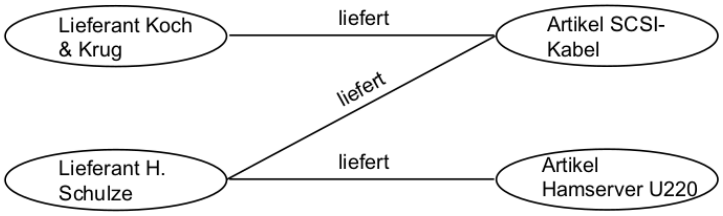

Abbildung 2.2: Entitäten und Beziehungen auf Instanzebene

Typebene

Bei der Typebene geht es um die bereits erwähnten Entity- und Beziehungs-Typen.

Unter einem Entitytyp versteht man die Menge der Entitäten einer Datenbank. Diese werden wie in Abbildung 2.3 durch Rechtecke dargestellt. Es kann ebenfalls zu Überschneidungen von Entitytypen kommen, was im Fachjargon durch die Eigenschaft „nicht disjunkt" beschrieben wird. So können zum Beispiel bei der Modellierung eines Gemischtwarenladens, Entitäten vom Entitytyp „Angestellter" ebenfalls im Typ „Kunde" enthalten sein.

Bei den Beziehungs-Typen kommt das selbe Prinzip zu tragen, wie bei den Entitytypen, bis auf den Unterschied, dass es hier keine Überschneidungen zwischen den Typen geben kann. Wie bereits in Abbildung 2.3 ersichtlich, wird ein Beziehungs-Typ durch eine Raute

im ER-Diagramm dargestellt.

Das letzte grundlegende Element eines ER-Diagramms ist das Attribut. Ein Attribut wird durch Ellipse dargestellt und ist dem jeweiligem Entity- bzw. Beziehungs-Typen zugeordnet um dessen Eigenschaften anzugeben. Durch sie wird es möglich den jeweiligen Typ zu klassifizieren, charakterisieren und identifizieren.

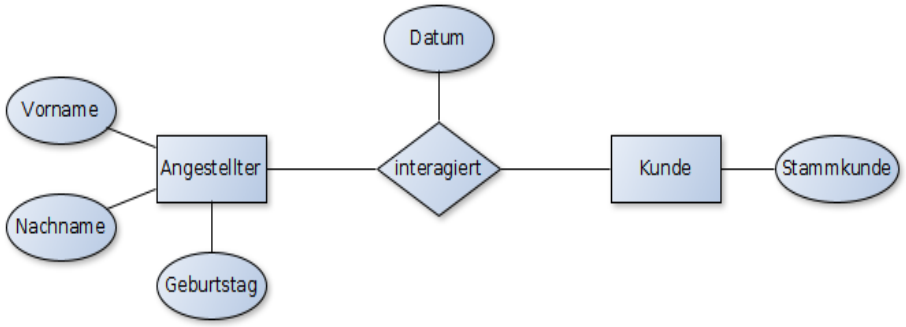

Abbildung 2.3: Darstellung von Attributen, Entity- und Beziehungs-Typen

2.2.2 Beziehungstypen

→*Passet*

Kardinalitäten

Unter der Kardinalität eines Beziehungstypen versteht man die quantitative Beschreibung mit Hilfe der (1,M,N)-Notation. Durch dieses Modell erhält man die Information wie viele Entitäten des einen Entitytypen mit wie vielen Entitäten des anderen Entitytypen maximal in Beziehung stehen können. Dabei wird jede Beziehung in zwei gerichtete Teile aufgeteilt, damit diese getrennt betrachtet werden können.

Dem Namen entsprechend kommen bei der Notation als Maximalwerte viele, sprich N bzw. M oder 1 in frage woraus sich folgende Kombinationen bei den Beziehungstypen ergeben:

- 1:1
- 1:N bzw. N:1
- M:N

Wendet man dieses Modell auf das Beispiel der Abbildung 2.3 an, ist eine Kardinalität von N:M in Betracht zu ziehen.

(min,max)-Notation

Bei der (min,max)-Notation handelt es sich um eine Erweiterung der (1,M,N)-Notation, denn durch das Hinzufügen der 0 in den Wertebereich, ist es dem Nutzer möglich ein Minimum für die Beziehungstypen zum Ausdruck zu bringen.

Dargestellt werden die Angaben als Intervalle. Außerdem wird bei dieser Notation das „N" in seiner grundlegenden Form durch einen „*" dargestellt. In der nachstehenden Abbildung finden Sie das vorangegangene Beispiel beschrieben durch die (min,max)-Notation.

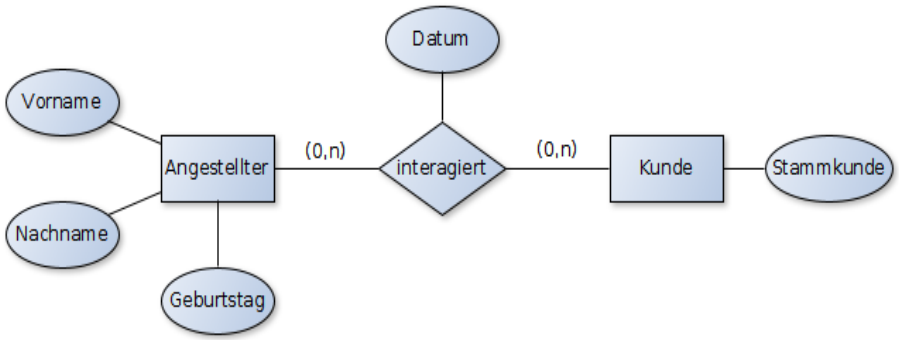

Abbildung 2.4: ER-Diagramm beschrieben durch (min,max)-Notation

2.2.3 Attribute

→*Passet*

Im Grunde sind Entitytypen und Attribute gleich. Deshalb stellt sich die Frage, wann man eine Informationseinheit im ER-Modell als Attribut oder Entitytyp modelliert. Die folgenden Merkmale helfen bei der Einordnung:

- Das Attribut ist immer einem Entitytyp zugeordnet, welcher eigenständig ist.
- Der Entitytyp wird durch seine Merkmale definiert. Attribute wiederum durch den erhalten Wert und Namen.

Die Attribute werden in zwei Kategorien aufgeteilt. In Schlüssel- und Nichtschlüsselattribute. Ein Schlüssel wird im Buch „**taschenbuch**" folgend definiert:

> Eine Attributmenge wird als **Schlüssel** bezeichnet, wenn sie eine eindeutige Identifizierung eines Entities eines Entitytyps ermöglicht und minimal ist.

Der für die Identifizierung verwendete Schlüssel wird Primärschlüssel (Primary Key) genannt und üblicherweise unterstrichen um ihn hervorzuheben. Sollte man keinen eindeutigen, minimalen Schlüssen finden wird ein „künstlicher" Schlüssel verwendet. Diese sind meist Zahlen die pro weiteren Entity um eins erhöht werden.

Die bereits erwähnten Nichtschlüsselattribute sind jene die nicht als Schlüssel geeignet sind und deshalb rein der Charakterisierung dienen. Bei der Zuordnung der Attribute sollte man, für die Vermeidung von Redundanzen, beachten, dass das Merkmal innerhalb des Schemas eindeutig ist und nicht in mehreren Entities, bezüglich auf den Wert des Merkmales, zugeordnet ist.

2.3 Andere Vertreter semantischer Datenmodelle

→*Passet*

Neben dem Entity-Reltionship-Modell von Chen sind während den 70er und 80er Jahren noch weitere semantische Datenmodelle entstanden. Abgesehen vom „Relational Model/Tasmania" von Codd, und dem „Functional Data Model" von Kerschberg gibt es noch das folgende Modell:

SDM (Semantic Data Model)

- Wurde von Hammer und McLoed 1981 erfunden.
- Laut dem Buch **semantischOnlineBuch** befolgt es folgende Grundsätze:

    1. To serve as a formal specification mechanism for describing the meaning of a database.
    2. To provide a documentation and interaction medium for database users.
    3. To provide a high-level, semantic based front-end user interface to a database.
    4. To serve as a foundation for supporting effective and structured design of databases.

Teil II

Methodik

Kapitel 3

Methoden

Quellenangaben?

3.1 Python

→*Fischbacher*

3.1.1 Geschichte

Die im Jahre 1991 von Guido van Rossum, einem niederländischen Software-Entwickler, veröffentlichte Programmiersprache Python, wurde ursprünglich für das Betriebssystem Amoeba entwickelt und sollte die Programmier-Lehrsprache ABC ablösen. Mit der ersten Vollversion, die unter dem Namen Python 1.0 im Jahre 1994 erschienen ist, wurde das Konzept der funktionalen Programmierung implementiert. Im Jahre 2000 erschien Python 2.0 mit einer voll funktionsfähigen Garbage Collection, sowie die Unterstützung für den Unicode-Zeichensatz. Python 3.0 wurde 8 Jahre später am 3.Dezember 2008 veröffentlicht. Mit der Version 3.0 kamen tiefgreifende Änderungen an der Sprache, die dazu führten, dass sich die Python Software Foundation dafür entschied, Python 2.7 und Python 3.0 bis Ende 2019 parallel mit neuen Versionen zu unterstützen. Die neuste Version die bis Heute erschienen ist, ist Version 3.7, die am 27.Juni 2018 erschienen ist.

Quelle? Fachlic

3.1.2 Idee und Zweck

Python ist eine sehr übersichtliche und einfache Programmiersprache und war in erster Linie dafür gedacht, das Programmieren zu erlernen. Das wird erzielt, in dem Python mit wenigen Schlüsselwörtern auskommt und die Syntax, im Vergleich zu anderen Programmiersprachen, sehr reduziert ist. Außerdem ist die Standartbibliothek von Python überschaubar und leicht erweiterbar, was zu folge hatte, dass heute etliche Bibliotheken zu Verfügung stehen.

3.1.3 Verwendung

Da Python über eine so große Vielfalt an Bibliotheken verfügt, einfach zu Programmieren ist und der Code sehr übersichtlich ist, bot es sich für dieses Projekt hervorragend an. Beispielsweise für die Implementierung der Umsetzung mittels Graphviz, die durch die von Graphviz zu Verfügung gestellten Bibliothek mit dem gleichen Namen programmiert wurde. Ein anderes Beispiel wo dieses Projekt von Python profitiert hat wäre die Eigenschaft von Python, dass es möglich ist, Python-Programme als Module in anderen Sprachen einzubetten.

3.2 Pycharm als IDE

Quellenangaben?

→*Fischbacher*

3.2.1 Allgemeines

3.2.1.1 Version

PyCharm ist eine IDE vom Unternehmen JetBrains die im Juli 2010 erschienen ist und viele Eigenschaften sowie Features mit den anderen IDEs von JetBrains teilt. Dabei hat JetBrains wie der Name PyCharm schon andeutet die IDE explizit für die Programmiersprache Python entwickelt. Die aktuellste Version ist die Version 2018.3.5, die am 27. Februar 2019 erschienen ist. Jedoch ist die neue Version 2019.1 schon in Entwicklung und wird noch im zweiten Quartal von 2019 verfügbar sein. In der Regel werden jedes Jahr um die 3 Versionen veröffentliche, jedoch handelt es sich dabei eher um kleinere Updates oder Hotfixes als um große Erneuerungen.

Zu ausführlich, d

3.2.1.2 Editionen

Außerdem gibt es PyCharm in drei verschiedenen Varianten, einmal die PyCharm Community Edition, die PyCharm Professional Edition und die PyCharm Educational Edition.

- Die Community Edition ist Open-Source und damit gratis für jeden erhältlich.
- Die Professional Edition verfügt über mehr Features, man benötigt für diese Version jedoch eine Lizenz die man kaufen muss.Für dieses Projekt wurde die Professional Edition verwendet, da JetBrains die Lizenzen gratis für Schulen anbietet.
- Die dritte Edition ist zum erlernen von Python gedacht.

Bla, bla.

3.2.2 Funktionen

3.2.2.1 Intelligenter Code Editor

PyCharm bietet intelligente Code Vervollständigung, Syntax hervorheben, automatische Code Refactorings. Die IDE erkennt copy/pasted Code und macht refactor Vorschläge.

3.2.2.2 Web Development Frameworks

PyCharm supportet auch spezifische Web Development Frameworks wie z.B.:

- Django
- Flask
- Google App Engine
- web2py

Relevanz für DA?

3.2.2.3 Cross-technology Development

Neben Python unterstützt die IDE auch noch andere Sprachen wie z.B.: JavaScript, TypeScript, SQL und HTML/CSS.

3.2.2.4 Built-in Developer Tools

Debuggen, Testen und Profiling ist mit PyCharm auch möglich und wird durch ein GUI für jeden einfach zu bedienen. Automatisches deployment auf einem remote host kann einfach eingestellt werden und außerdem bietet die IDE eine schnelle und benutzerfreundliche GUI für Versionsverwaltungsprotokolle wie z.B.: Git, Mercurial und SVN.

Werbeprospekt? Relevanz für DA?

3.2.3 Vorteile

- Ein Vorteilen der IDE ist der ~~hervorragende~~ Support der mittels Forum oder Mail Kontakt direkt mit den Entwicklern erfolgt.
- Neben dem Support wird bei JetBrains auch die Benutzerfreundlichkeit ihrer IDE ~~groß geschrieben.~~ Die einzelnen Funktionen sind übersichtlich von den anderen getrennt und in den Untermenü mit gut erkennbaren Symbolen gekennzeichnet.
- Der WSL Interpreter war bei dem Projekt eine große Hilfe, da nicht jeder auf dem selben Betriebssystem arbeitete. Dadurch konnte man selbst, wenn auf einem Rechner mit Windows gearbeitet wurde, in einer Linux Umgebung testen.
- Durch die große Community von PyCharm und JetBrains gibt es für jede IDE eine Vielzahl an Plugins, die nicht nur das Programmieren vereinfachen, sondern auch die Versionsverwaltung und andere, für die Entwicklung wichtige, Vorgänge.
- Durch die REPL Python ~~console~~ wird dem Entwickler das Testen ~~stark~~ vereinfacht, da man direkt von der IDE aus das Programm starten, testen und debuggen kann. Außerdem wird einem der derzeitige Zustand seiner Variablen angezeigt.

3.3 Git CLI

3.3.1 Verwendung

3.3.2 Vorteile der CLI

3.4 XML

3.4.1 Allgemeines zu XML

3.4.1.1 Definition

Das Datenformat *Extensible Markup Language (XML)* ist mittlerweile weit verbreitet und dient häufig als Basis für viele Technologien. XML wird im Buch „**Taschenbuch Datenbanken**" wie folgt definiert:

> XML wurde mit der Zielsetzung des erleichterten Datenaustausches als Vereinfachung seines Vorgängerstandards *SGML* eingeführt. Inzwischen ist es aber viel mehr als das: Es bildet beispielsweise die Grundlage vieler Technologien der serviceorientierten Architektur. XML wird dabei zur Definition von Sprachen verwendet und legt gleichzeitig die Basissyntax für diese Sprachen fest.

3.4.1.2 Aufbau einer XML Datei

Der Aufbau einer *XML-Datei* erinnert an den einer *HTML-Datei*, da der Code ebenfalls mit Tags versehen ist. Ein großer Unterschied besteht jedoch darin, dass die Tags von Elementen in XML selbst wählbar und nicht vordefiniert sind. Die Daten werden in den Tags gespeichert.

In einem *XML-Document* befindet sich der Code der eigentlichen Daten, jedoch kann auch ein Kopf angegeben werden. Dies ist jedoch optional. In diesem Kopf befinden sich Informationen zu der Version des Datenformats sowie über die Attribute *encoding* und *standalone*.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

- *version* gibt die aktuelle Version an, welche in dem Dokument verwendet wird. Im Moment ist der Wert immer 1.0, da noch keine weiteren Versionen von *XML* veröffentlicht worden sind.
- *encoding* gibt an, welche Kodierung der Zeichen benutzt werden soll. Standardmäßig gilt *UTF-8* als verwendete Kodierung.
- *standalone* gibt an, ob auf eine externe DTD zugegriffen werden muss, um korrekte Werte für bestimmte Teile zu ermitteln. Der Standardwert für dieses Attribut ist mit *no* definiert.

Sowohl *encoding* als auch *standalone* stellen optionale Attribute dar. Falls diese nicht explizit angegeben werden, gelten die zuvor erwähnten Standardausprägungen.

Im restlichen Dokument können dann beliebig viele Tags erstellt werden. Jedoch gilt zu beachten, dass *XML* eine gewisse Syntax verlangt. Unter anderem muss jeder Datensatz über einen Start-Tag und einen End-Tag verfügen.

```
<Autor>
    <name> Andreas Prinz </name>
</Autor>
```

Bei der Namensgebung der Tags ist zu beachten, dass Start- und End-Tag gleich heißen. Groß- und Kleinbuchstaben sind in beliebigem Ausmaß erlaubt. Die Namen der Elemente und Attribute zwischen den Tags dürfen Unterstriche, Bindestriche, Punkte und alphanumerische Zeichen enthalten.

```
<Autor>
  <name> Andreas Prinz </name>
  <geb_datum> 01.11.1999 </geb_datum>
</Autor>
```

Weiters können Elementen in *XML* Attribute hinzugefügt werden. Dabei ist zu beachten, dass eine Eigenschaft nur bei dem Start-Tag eingefügt werden kann. Der Wert wird dem Attribut mittels einem Gleichheitszeichen zugewiesen. Der Inhalt muss jedoch von einfachen oder doppelten Anführungszeichen eingeschlossen sein. Ein Beispiel für ein Element mit einer Eigenschaft sieht wie folgt aus:

```
<diplomarbeit schuljahr="2018/19">
   <Autor> Andreas Prinz </autor>
<diplomarbeit>
```

In einem XML-Dokument können Namensräume vergeben werden. Diese gestalten die Datei in einem übersichtlicheren Format. Es besteht jedoch keine Pflicht, Namensräume zu verwenden, da sie optional sind. Häufig tritt der Fall ein, dass verschachtelte Namensräume zum Einsatz kommen, um unterschiedliche Teile einer Anweisung besser differenzieren zu können.

3.4.1.3 Gültigkeit von XML-Dokumenten

Gültige XML-Dokumente entsprechen dem XML-Standard und verfügen über eine *DTD*. Unter einer Dokumenttyp-Definition (DTD) wird eine Grammatik für eine XML-Datei verstanden. In diesem Dokument befinden sich alle Elemente, Attribute und Entities, welche in dem XML-Code verwendet werden dürfen. Weiters beinhaltet eine *DTD* den Kontext, in dem die Elemente auftauchen. Eine weitere Eigenschaft einer *DTD* besteht darin, dass es die Möglichkeit gibt, die Anzahl eines Elementes an einer bestimmten Stelle festzulegen.

Dafür stehen 3 Operatoren zur Verfügung:

- '?' Das Element darf maximal einmal vorkommen
- '+' Das Element muss mindestens einmal vorkommen
- '*' Das Element kann beliebig oft vorkommen

Neben *DTD* verfügt XML auch über ein Schema, mit dem weitere Regeln definiert werden können. Weiters kann mit Hilfe dieses Schemas ein XML-Dokument auf seine Gültigkeit überprüft werden. Ein großer Unterschied zu einer *DTD* liegt darin, dass das XML-Schema ebenfalls eine Datei vom Typ XML darstellt und über kein eigenes Datenformat verfügt.

Quelle: http://wwwlgis.informatik.uni-kl.de/archiv/wwwdvs.informatik.uni-kl.de/courses/seminar/WS02

3.4.2 XERML

3.4.2.1 Verwendungsgrund

3.4.2.2 Aufbau

Ein Beispiel für den Aufbau einer *XERML-Datei* anhand des modellierten Datenmodelles *Fußball* (nur Teile des Datenmodelles abgebildet):

```
<?xml version="1.0" encoding="utf-8"?>

<erm version="0.2">

<!-- Front Matter -->

<title name="Fussball"/>
<title name="soccer" lang="en"/>

<!-- Entity-Types -->

<ent name="mannschaft">
    <attr name="name" prime="true"/>
    <attr name="gründungsjahr"/>
    <attr name="adresse"/>
</ent>

<ent name="spieler">
    <attr name="spielposition"/>
</ent>

<ent name="spiel">
    <attr name="spielort" prime="true"/>
```

```
    <attr name="datum"    prime="true"/>
    <attr name="mannschaft_heim"/>
    <attr name="mannschaft_ausw"/>
    <attr name="schiedsrichter"/>
    <attr name="ergebnis"/>
</ent>

<!-- Relationship-Types  -->

<rel to="spielt bei">
    <part ref="spieler" min="1" max="1"/>
    <part ref="mannschaft" min="1" max="n"/>
</rel>

<rel to="spielt mit bei">
    <part ref="mannschaft" min="1" max="n"/>
    <part ref="spiel" min="1" max="n"/>
</rel>

</erm>
```

3.4.2.3 Vergleich mit XML

3.5 XML

3.5.1 Entstehung

3.5.2 Verwendungsmöglichkeiten

3.5.3 XERML

3.5.3.1 Verwendungsgrund

3.5.3.2 Aufbau

3.5.3.3 Vergleich mit XML

»»»> b97800223f14b3743b824935088eb2e078256abb

3.6 PIC-Code

3.6.1 Definition

Die Sprache PIC-Code ist eine Erweiterung von 'troff'. Unter dem Begriff 'troff' versteht man ein Textsatzsystem, welches dem Benutzer erlaubt einen qualitativ hochwertigen Text zu erstellen oder ein Diagramm zu zeichnen, wofür jedoch Erweiterungen benötigt werden.[1] PIC stellt zusätzlich Features zur Verfügung, mit welchen zum Beispiel Boxen, Linien oder Ellipsen einfach dargestellt und verbunden werden können.

[1] **https://de.wikipedia.org/wiki/Troff**.

Falls in einem troff-Dokument beispielsweise ein Diagramm gezeichnet werden soll ist dies ohne weiteren Extras nicht möglich. Um dies umzusetzen kann PIC in den Code eingebunden werden. Mithilfe der Erweiterung sollte es nun möglich sein, das gewünschte Objekt zu generieren, da PIC über eine mittlere Anzahl an Möglichkeiten bietet, Gegenstände zu zeichnen, zu verbinden oder sogar zu färben.

3.6.2 Aufbau einer PIC Datei

Der Aufbau von jedem PIC Programm ist im Prinzip gleich. Jede Datei muss zu Beginn des Codes ein '.PS' stehen haben, da dies dem Compiler kennzeichnet, dass ab diesem Zeitpunkt Befehle in der Sprache PIC zu erwarten sind. Falls der Eintrag '.PS' fehlt, so wirft der Compiler während des Ausführens einen Syntax-Fehler aus und weist darauf hin, dass der erforderliche Ausdruck fehlt.

Das Ende eines PIC Programmcodes ist mittels '.PE' gekennzeichnet. Das Fehlen dieses Eintrages trägt dazu bei, dass der Compiler kein Ende der Datei vorfindet und es tritt ebenfalls ein Syntaxfehler auf.

Der restliche Inhalt der Datei zwischen Anfang und Ende kann beliebig aufgebaut sein, solange er syntaktisch richtig ist.

Beim Kompilieren einer Datei wird am Beginn des Dokuments gestartet, von dort wird Zeile für Zeile ausgeführt, bis der Compiler den Befehl '.PE' erkennt. Falls sich nach diesem Ausdruck noch weiterer Code befindet, dann endet der Compiler ebenfalls mit einem Syntaxfehler. Wenn kein weiterer Fehler aufgetreten ist, kann das kompilierte Programm nun ausgeführt oder in ein ausgebbares Format konvertiert werden.

Es reicht jedoch nicht, den Code einfach abzuspeichern und auszuführen, da das Programm im Normalfall als einfaches Textdokument abgespeichert wird und nicht ausführbar ist. Dieses kann nicht kompiliert oder angezeigt werden. Deswegen muss das geschriebene Programm mittels dem Befehl *'groff -p File.pic > File.pdf'* kompiliert werden. Um das generierte Diagramm anzeigen zu können wird der Inhalt des Textdokuments in eine PDF Datei konvertiert, mit welcher das erzeugte Diagramm angezeigt werden kann.

Ein Beispiel für einen Programmcode könnte so aussehen:

```
.PS
ellipse "document";
arrow;
box "\fIpic\fP(1)"
arrow;
box width 1.2 "\fIgtbl\fP(1) or \fIgeqn\fP(1)" "(optional)" dashed;
arrow;
box "\fIgtroff\fP(1)";
arrow;
ellipse "PostScript"
.PE
```

Abbildung 3.1: Beispiel eines PIC-Programmcodes[2]

Aus diesem Codebeispiel wird dann folgender Graph generiert:

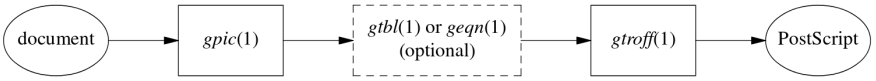

Abbildung 3.2: Erzeugter Graph von dem Beispielcode

3.6.3 Objekte zeichnen in PIC

In PIC-Code ist es möglich verschiedene Objekte zu zeichnen. Hierbei gibt es vordefinierte Objekte, jedoch ist es möglich auch Objekte zu generieren welche nicht seit Anfang an mit implementiert sind.

Zu den grundsätzlichen Objekten von PIC zählen:

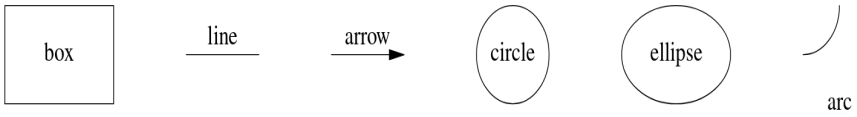

Abbildung 3.3: Vordefinierte Objekte

Die Form einer Raute ist standardmäßig nicht definiert, lässt sich jedoch mittels geringem Aufwand leicht implementieren. Es steht nämlich die Möglichkeit zur Verfügung, einfach den Rand der Raute mittels verbundenen Linien zu zeichnen:

```
1  .PS
2  box invis;
3  line from last box .n to last box .e then to last box.s then to last box .w then to last box .n
4  .PE
```

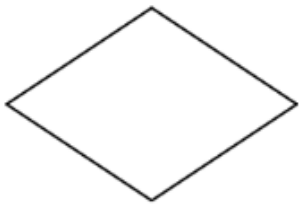

Abbildung 3.4: Code zum Zeichnen einer Raute und das Ergebnis

PIC verfügt über eine große Auswahl an Möglichkeiten, um ein bestimmtes Objekt zu zeichnen. Man kann durch zusätzliche Argumente zum Beispiel die Breite und die Höhe einer Box festlegen, ob der Rahmen eine durchgezogene Linie oder nur strichliert dargestellt werden soll oder ob die Box in einer bestimmten Farbe zu generieren ist.

Der benötigte Code um eine Ellipse in der Farbe gelb zu zeichnen sieht wie folgt aus:

Abbildung 3.5: Code zum Zeichnen einer Ellipse in der Farbe gelb und das Ergebnis

Ein vorteilhaftes Feature von PIC-Code ist, dass man mit Labels arbeiten kann. Darunter versteht man, dass Objekten eine bestimmte ID zugewiesen werden kann, um im nachhinein einfach darauf zuzugreifen. Dies kann unter anderem nützlich sein wenn 2 bestimmte Elemente verbunden werden sollen, welche nicht hintereinander gezeichnet wurden. Dabei sind gewisse Syntaxregeln für ein Label einzuhalten. Es darf nämlich keine Leerzeichen enthalten und muss mit einem Großbuchstaben beginnen. Weiters dürfen keine Ziffer und Umlaute vorkommen. Falls jedoch eines der aufgezählten Kriterien nicht erfüllt ist, so tritt beim Kompilieren der Datei ein Syntaxfehler auf.

Um das Layout der gezeichneten Elemente zu bestimmen gibt es mehrere Methoden. Wenn keine zusätzlichen Befehle angegeben werden generiert PIC das erste Objekt in der linken oberen Ecke des PDFs und fügt die restlichen rechts daneben ein. Es gibt jedoch die Möglichkeit das Layout der generierten Datei mittels addieren oder subtrahieren der Koordinaten zu verändern und somit den Standort der Graphen zu bestimmen. Eine weitere Art der Spezifizierung kann dadurch erreicht werden, indem x und y Werte direkt in den Befehl eingebunden werden.

3.6.4 Erstellen einer PIC-Datei

Zu Beginn muss erstmals eine Datei erstellt werden, in welche der generierte PIC-Code zu schreiben ist. Dabei ist egal, ob der Typ des Dokuments ein Textdokument mit der Endung *.txt* oder eine PIC-Datei mit der Endung *.pic* ist. Sobald diese erstellt ist, sind der Beginn und das Ende des Codes mit den Schlüsselwörtern *.PS* und *.PE* zu kennzeichnen. Der Inhalt kann mittels einfachen Zeichenketten in die Datei eingefügt werden.
Sobald in das erstellte Dokument der PIC-Code ergänzt wurde, kann die Datei kompiliert

```
.PS
A: box;
move;
B: circle;
down;
move;
C: ellipse;
line from A to C;
.PE
```

Abbildung 3.6: Objekte mit Labels versehen und verbinden

```
.PS
A:box at (100, 100);
.PE
```

Abbildung 3.7: Zeichnet eine Box bei den Koordinaten (100,100)

```
def write_erd_to_file(pic, inputfile):
    root = xerml_einlesen.get_root(inputfile)
    if os.path.exists("erd.pic"):
        os.remove("erd.pic")
    file = open("erd.pic", "w")
    file.write(pic)
    file.close()
    print('Your file is created at: ' + os.getcwd() + '/erd.pic')
```

Abbildung 3.8: PIC-Datei erstellen und den fertigen Code in das Dokument einfügen.

und angezeigt werden.

3.6.5 Vorteile und Nachteile von PIC

3.6.5.1 Vorteile

Die Erweiterung PIC bietet einige Vorteile, welche für die Generierung von Dokumenten und Graphen hilfreich sein können.

- Da die Sprache sehr einfach aufgebaut ist und die Namen der Befehle bzw. Objekte sprechend gewählt wurden, fällt es einem späteren Leser nicht schwer den Code zu verstehen und zu verwerten. Vor allem mit Hilfe der vordefinierten Objekte wird das Generieren grundlegender Graphen um einiges vereinfacht und erfordert keinen großen Aufwand.

```
.PS
box "from"
move 0.75;
ellipse "to"
arc cw from 1/3 of the way \
    between last box .n and last box .ne to last ellipse .n;
.PE
```

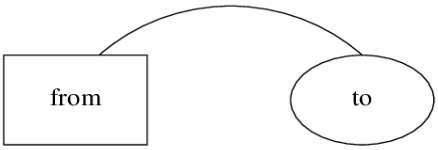

Abbildung 3.9: leicht lesbarer Code mit sprechenden Objektnamen

- Weiters kann man dank der Fehlermeldungen und des leicht zu lesenden Codes sowohl syntaktische als auch logische Fehler ohne großem Aufwand finden und beheben. Dies setzt jedoch voraus, dass sich der Entwickler mit der Programmiersprache auseinander gesetzt hat, da man von den Fehlermeldungen nur herauslesen kann, an welcher Stelle sich der nicht korrekte Ausdruck befindet und welches Symbol diesen auslöst. Über die Richtigstellung wird jedoch keine Information angegeben.

```
.PS
box. "Syntaxfehler"
.PE
```

```
[andi@andreas-pc  ]$ groff -p Label.txt > Label.pdf
pic:Label.txt:2: syntax error before `.'
pic:Label.txt:2: giving up on this picture
[andi@andreas-pc  ]$
```

Abbildung 3.10: Fehlerhafter Code und Ausgabe eines Syntaxerrors

3.6.5.2 Nachteile

Neben den Vorteilen gibt es jedoch auch Nachteile, welche im Vergleich zu den Pro-Argumenten überwiegen. Zu den negativen Aspekten zählen:

- Die Anzahl der standardmäßig definierten Objekte in PIC ist nur gering. Daraus folgt, dass der Aufwand andere Objekte zu zeichnen steigt, auch wenn das nur geringfügig bzw. der Code leicht zu generieren ist. Trotz all dem werden die Umstände

das gewünschte Ergebnis zu bekommen erschwert.

- Weiters ist über PIC-Code zu sagen, dass das Verbinden von zwei unterschiedlichen Objekten in der Theorie recht einfach ist, praktisch jedoch nur mit viel Aufwand verbunden umsetzbar ist. Es müssen nämlich die Richtungen (Norden, Osten, Süden, Westen) angegeben werden, denn ohne diese Information befindet sich der Beginn und das Ende der gezeichneten Linie in der Mitte des jeweiligen Objektes. Falls dieser Gegenstand beschriftet ist hat dies zur Folge, dass der Inhalt der Box schwerer bis nicht mehr lesbar wird.

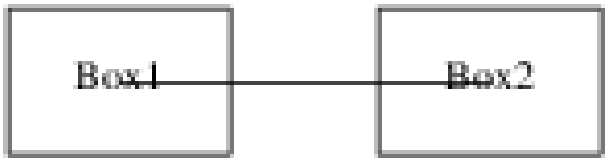

Abbildung 3.11: Connectoren ohne zusätzlicher Information über Start- und Endpunkt

- Die Sprache PIC zu erlernen ist nicht unbedingt einfach. Man benötigt eine Menge an Zeit um sich das Wissen anzueignen. Der Grund dafür ist, dass es über PIC-Code nur wenige Dokumentationen gibt. In diesen Dokumenten wird die Erweiterungssprache jedoch nur grob erklärt beziehungsweise werden spezifische Anwendungsfälle nicht erwähnt und somit fehlen viele Informationen, die in der Praxis vorteilhaft wären. Das Wissen muss man sich schrittweise durch eigenes Probieren und Testen aneignen.

- Bei der Generierung des Layouts treten häufig Probleme auf. Das liegt daran, dass PIC die Koordinaten in *inches* berechnet, jedoch eine PDF Datei mit Pixel arbeitet. Dies erschwert den Zugriff mittels absoluten Koordinaten, da von PIC nicht überprüft wird, ob dieser Standort noch innerhalb der Werte liegt oder ob dieser ausserhalb des gegebenen Bereiches gezeichnet wird. Dies kann zur Folge haben, dass ein Objekt zu einem gewissen Teil abgeschnitten wird wie in der Abbildung 3.11 zu sehen ist.

```
.PS
box at (0, 300);
.PE
```

Abbildung 3.12: Box wird wegen zu großen Koordinaten abgeschnitten

Wenn die Koordinaten zu groß werden, kann es auch zu dem Fall kommen, dass diese nicht einmal mehr in der Nähe der maximalen Grenzen liegen. Das hat zur Folge, dass das Element nicht mehr angezeigt werden kann.

```
.PS
box at (500, 0);
boxwid=35;boxht=15;; box at (0, 0);
line from 1st box to last box;
.PE
```

Abbildung 3.13: Box wird wegen zu großen Koordinaten ausserhalb des Sichtfelds gezeichnet

3.6.6 Ist PIC-Code für das Projekt geeignet?

3.6.6.1 Generierung der Objekte des ERD's

PIC verfügt über viele einfache Mittel um ein Entity Relationship Diagram zu generieren. Dank den bereits vordefinierten Objekten lässt sich das Zeichnen von dem Grundgerüst des ERD's relativ leicht gestalten. Bei der Implementierung einer Raute, welche PIC von Beginn an nicht bekannt ist, besteht ebenfalls kein großer Aufwand wie in Abbildung 3.4 bereits erwähnt wurde.

3.6.6.2 Zeichnen der Beziehungen

Weiters vereinfacht das Arbeiten mit Labels die Implementierung der Verbindungen zwischen den Elementen. Zuerst werden alle Boxen, Ellipsen und Rauten gezeichnet und sobald alle Elemente erstmals vorhanden sind, werden die Verbindungen zwischen den Objekten gezeichnet. Dank den Labels ist dies kein Problem mehr, da auf die verschiedenen Arten der Boxen leicht zugegriffen werden kann (siehe Abbildung 3.6).

3.6.6.3 Layout

Jedoch ist das Layout des ganzen Diagrammes ein großes Problem. Mittels PIC-Code war es nicht möglich, ein gut aussehendes ERD zu erzeugen. Daher musste mit Hilfe von *Graphviz* das Layout im vorhinein generiert werden, wobei nur die Koordinaten der Knoten erzeugt werden und nicht das ganze Diagramm. Ausgehend von diesen Werten können dann die Objekte einfach im PIC-Code positioniert werden.

Jedoch tritt dank dieser Methode ein weiteres Problem auf und zwar, dass man beim Generieren der Koordinaten die richtigen Parameter für den Befehl *"graphviz layout"* benötigt. Die korrekten Zusatzeinstellungen zu finden ist jedoch ziemlich schwer, da diese mit dem Dokument übereinstimmen müssen, da sonst die Werte der Koordinaten zu groß werden

```python
def nx_graph(rel):
    erd = nx.Graph()
    for curr in rel:
        if len(curr) > 1:
            for i in range(len(curr)):

                erd.add_edge(curr[0], curr[i])
        else:
            continue

    pos = graphviz_layout(erd, prog='neato', args="-Gsplines=true -Goverlap=false -Gsize=600,400! -Gdpi=1 -Gratio=fill")
    print(pos)
    return pos
```

Abbildung 3.14: Python Code zur Erzeugung der Koordinaten mittels Graphviz

und deswegen Elemente ausserhalb des Dokuments gezeichnet werden. Ausserdem erzeugt dieser Befehl die generierten Punkte in einem viel zu großen Ausmaß, sodass die Werte trotz richtigen Parametern noch bearbeitet werden müssen.

3.6.6.4 PIC-Kenntnisse aufbauen

Das Erlernen der Sprache PIC erweist sich als ein sehr zeitaufwendiger Vorgang, da es nur Dokumentationen gibt, die allgemeine Anwendungen beschreiben. Anhand von diesen Dokumenten ist es zwar möglich einen theoretischen Überblick über das Thema zu bekommen, jedoch nimmt es eine Menge an Zeit in Anspruch die Sprache auch in der Praxis umzusetzen. Sobald man das Prinzip von PIC verstanden hat, ist es nicht mehr schwer, dies auch anzuwenden, da es nicht viele Regeln gibt, die zu beachten sind.

3.6.6.5 Skalierung der Objekte

Die Elemente können in PIC leicht über das Argument *scale* skaliert werden. Dies ist hilfreich, wenn man weiß, wie groß das zu zeichnende Diagramm ist. Jedoch kann nicht automatisiert skaliert werden. Das heißt, dass die Größe des ERD's zuerst festgelegt werden muss. Da die Koordinaten im vorhinein schon erzeut werden, ist dies kein Problem, jedoch ist der Implementierungsaufwand größer.

3.6.6.6 Vergleich mit anderen Varianten

Im Vergleich zu den anderen Varianten Graphviz, Graphml und Libre Office Draw verfügt die Sprache PIC über weniger Methoden für die Implementierung und Generierung eines ERD's. Beispielsweise kann das erzeugte Diagramm im Nachhinein nicht mehr verändert werden, da es in einer *PDF-Datei* gespeichert wird. Bei den restlichen drei Methoden können einzelne Elemente nachträglich noch per Hand verschoben werden. Daher ist PIC im Vergleich zu den anderen Implementierungsvarianten am Wenigsten geeignet für dieses Projekt, jedoch ist es trotzdem mit einigem Aufwand möglich, ein ERD zu erzeugen.

3.7 yEd

→*Passet*

Das Programm yEd ist ein Graph Editor, mit dem die verschiedensten Diagrammtypen erstellt werden können. Die Applikation wurde von dem deutschen Unternehmen yWorks entwickelt und basiert auf der Java-Bibliothek yFiles. Dadurch ist punktet der Editor nicht nur mit mit den Layout-Algorithmen und den Analyse-Tools sonder auch mit einem übersichtlichen User-Interface, dass das erstellen und bearbeiten von Diagrammen vereinfacht.

3.7.1 Varianten und Installation

Um die Anwendung zu nutzen gibt es zwei Möglichkeiten. Die erste wäre den „yEd Live"-Dienst in Anspruch zu nehmen. Dabei handelt es sich um eine Online-Variante. Der Nachteil dieses Services ist, dass die Verarbeitung und auch Darstellung der Daten im Hintergrund anders ist als bei der Offline-Version. Dies führt unter anderem dazu, dass die erzeugten GraphMl-Dateien bei „yEd Live" nicht richtig bis hin zu gar nicht angezeigt werden.

Bei der zweiten Methode handelt es sich um die bereits erwähnte Offline-Version. Diese kann über die offizielle Internetseite von yWorks für das entsprechende Betriebssystem heruntergeladen und danach entsprechend installiert werden.

Wichtig ist es bei der Auswahl der Datei für die Installation darauf zu achten, ob diese auch die benötigte Java JRE beinhaltet und wenn nicht, ob man selbst auf dem Rechner die dazugehörige installiert hat. Wenn man sich dann im Installationsprozess befindet kommt man an einen Punkt bei dem man nach dem Installationspfad gefragt wird. Hierbei muss man darauf achten das Programm an einem Ort zu installieren zu dem man als Benutzer auch zugriff hat, weil das Command-Line-Tool bei der Erstellung eines ER-Diagrammes anbietet, die generierte Grafik, sofern es sich um eine GraphMl Datei handelt und man auf Linux arbeitet, direkt von der Kommandozeile heraus in yEd zu öffnen.

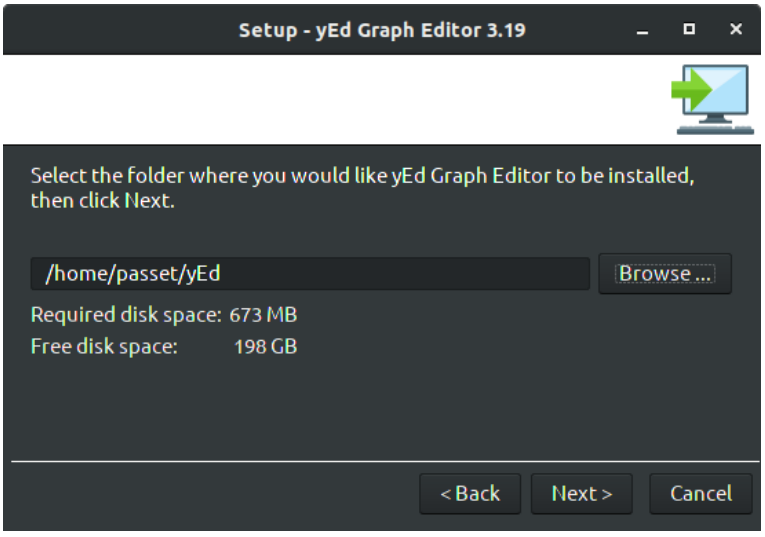

Abbildung 3.15: Angabe des Installationspfades im Installer von yEd 3.19

3.7.2 Oberfläche

→*Passet*

Das grafische Interface mit dem der Benutzer interagiert entspricht zwar nicht den modernsten Design Ideen aber bietet alles was man braucht. Grundlegend kann man die Oberfläche des Editors in 7 Bereiche aufteilen.

3.7.2.1 Palette

In der Palette befinden befinden sich die Elemente mit denen man einen Graphen erstellen kann. Diese Komponenten sind in Gruppen aufgeteilt um für einen besseren Überblick zu sorgen. Zu Beginn bietet der Editor einem die folgenden Gruppierungen mit den entsprechenden Elementen:

- Geometrische Knoten
- Moderne Knoten
- Kantentypen
- Gruppenknoten
- Swimlane- und Tabellenknoten
- Personen
- Computer-Netzwerk
- UML
- Flussdiagramm
- BPMN
- Entity Relationship
- SBGN
- Aktuelle Elemente

Es gibt auch für den Benutzer die Möglichkeit selbst eine Gruppe mit Elementen zu erstellen und diese dann auch in die Palette einzubinden.

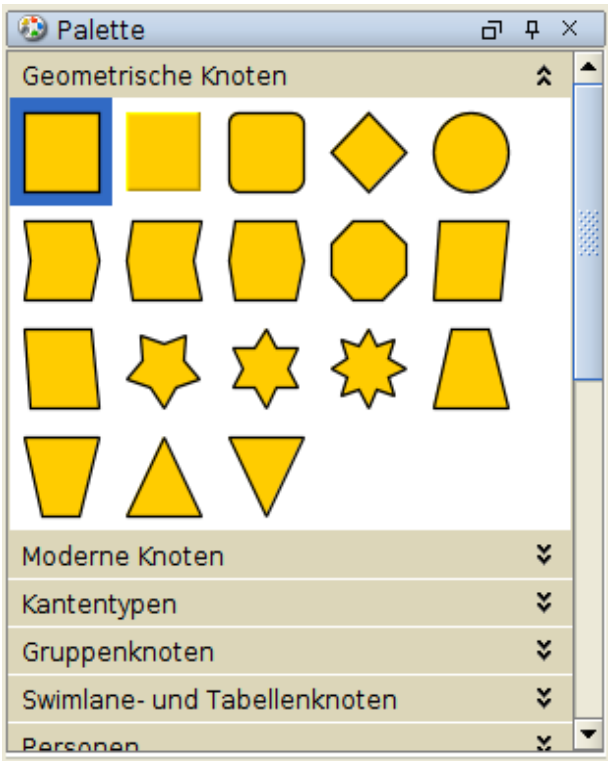

Abbildung 3.16: Palette in yEd bei Programmstart

3.7.2.2 Eigenschaften

→*Passet*

Der Bereich Eigenschaften zeigt die charakteristischen Werte den momentan ausgewählten Elementes an. Hier können allerlei verschiedener Einstellungen vorgenommen werden wie z.B. das ändern der Beschreibung, Größe oder auch Farbe, sofern diese änderbar sind. Die Merkmale in diesem Bereich werden in die Gruppen Allgemein, Beschriftung, Daten und eine für das Element spezifische Gruppe aufgeteilt. In der Abbildung 3.17 wäre dann die spezifische Gruppe „Entity Relationship".

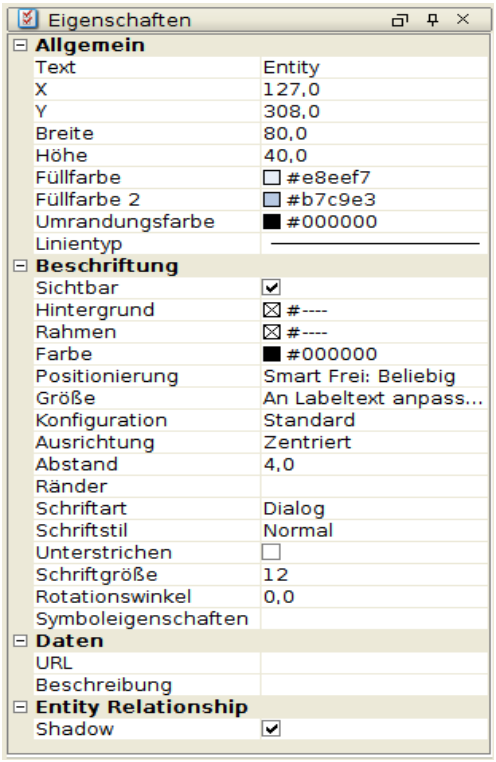

Abbildung 3.17: Eigenschaften eines Entitytypen in yEd

Sollte man kein Element seines Graphen ausgewählt haben, sieht man in diesem Bereich die Gesamtzahl an Knoten und Kanten.

3.7.2.3 Nachbarschaft

Auf der linken Seite der GUI befinden sich Bereiche die einem Auskunft über den Aufbau des aktuellen Graphen geben. Darunter befindet sich der Bereich Nachbarschaft. In diesem Fenster sieht man demnach alle Nachbar-Knoten des momentan ausgewählten Knoten. Zudem bietet es Registerkarten an in denen nur die Knoten innerhalb der Gruppe, Vorgänger oder Nachfolger angezeigt werden.

Dies kann sich bei einem Entity-Relationship Diagramm als großen Vorteil entpuppen, weil mit einem Klick herausgefunden werden kann mit welchen Beziehungstypen der Entitytyp in relation steht. In der Abbildung 3.19 wird die Nachbarschaft des Entitytypen Abteilung dargestellt.

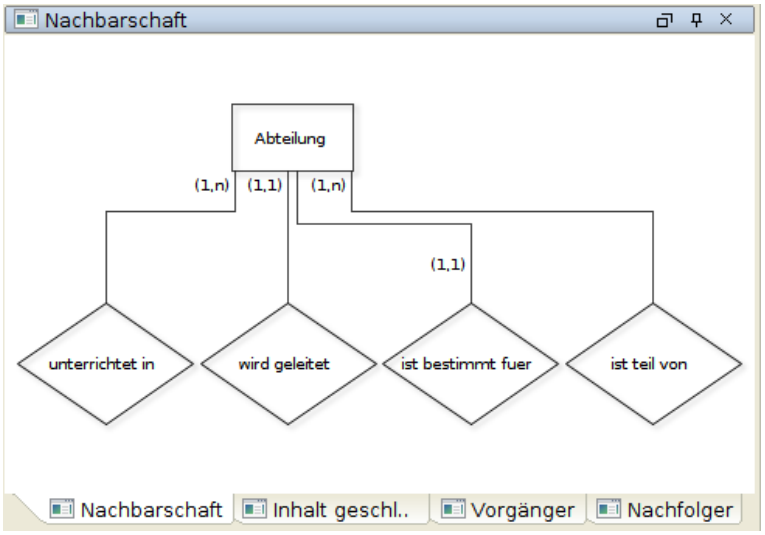

Abbildung 3.18: Nachbarschaft des Entitytypen Abteilung aus dem Datenmodell Schulinformationssystem

3.7.2.4 Struktur

→*Passet*

Im wesentlichen beinhaltet dieser Bereich eine Liste mit all den Elementen die sich zum momentanen Zeitpunkt auf der Hauptoberfläche von yEd befinden. Sie ist hierarchisch geordnet weshalb Teile des Graphen die zuerst erstellt wurden weiter oben angeführt sind als Teile die später hinzugefügt wurden.

Das Fenster Struktur ist eng mit den zuvor genannten Bereichen verbunden. Wählt man eine der Komponenten aus wird sie im Graphen markiert, die Eigenschaften dargestellt und die Nachbarschaft aufgezeigt.

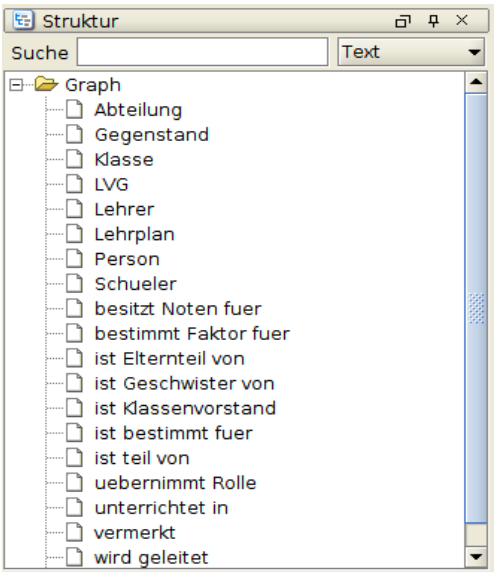

Abbildung 3.19: Strutkur des ER-Diagrammes Schulinformationssystem

3.7.2.5 Editor-Bereich

→*Passet*

In diesem Bereich wird der Graph dargestellt und durch Elemente aus der Palette ergänzt und verändert. Dies geschieht durch die Verwendung einer Maus oder durch drücken bestimmter Tasten auf der Tastatur. Die folgenden Aktionen und noch weiter nicht angeführte können auf diese Weise ausgeführt werden:

- Einen neuen Knoten erstellen und auswählen
- Eine Kante erstellen und auswählen
- Knoten verschieben
- Eine neuen Biegepunkt auf der Kante erstellen und verschieben
- Ein Element löschen
- Eine Beschriftung erstellen, auswählen, bearbeiten und verschieben

Die folgende Abbildung zeigt einen von yEd generierten Graphen den man ohne weiteres durch die Verwendung der eben genannten Aktionen erstellen kann.

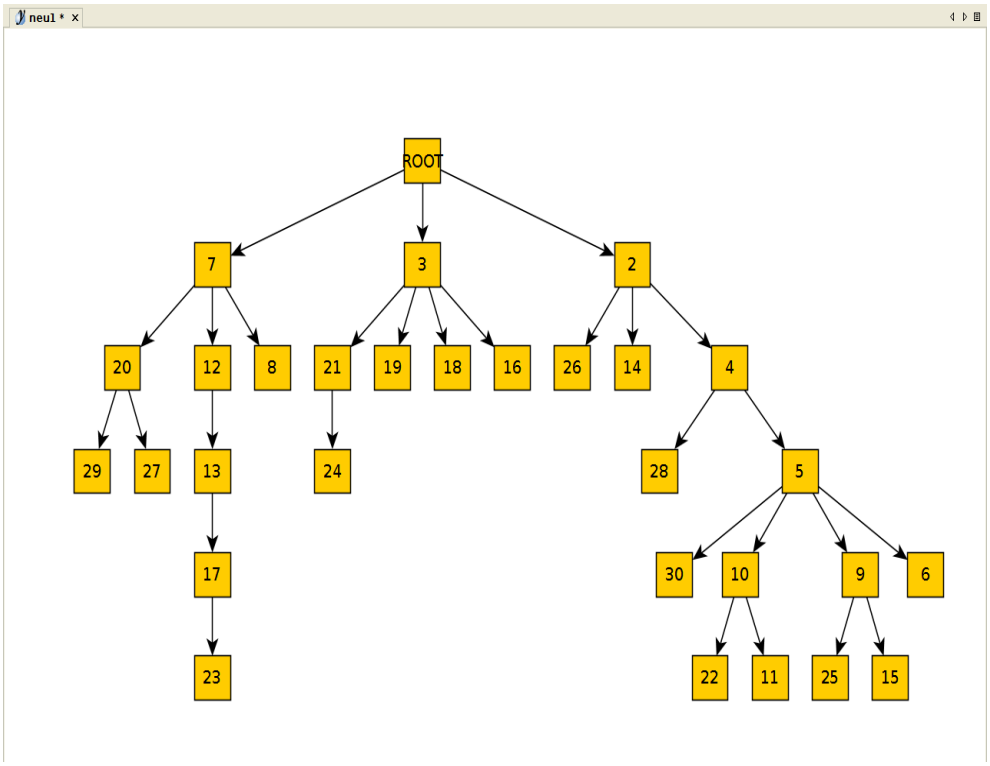

Abbildung 3.20: Beispiel Graph generiert von yEd

3.7.3 Editier-Hilfen

Um den Benutzer bei der Anordnung der Elemente seines Graphen zu unterstützen, bietet yEd 3 Features.

3.7.3.1 Snap Lines

→*Passet*

Im **yEdManual** wird das Grundkonzept der „Snap Lines" , zu deutsch Hilfslinien, wie folgt beschrieben:

> Snap Lines help you to interactively create a graph with an appealing layout.

Zusammenfassend lässt sich also sagen, dass die „Snap Lines" für ein ansprechendes Layout sorgen. Diese Linien werden angezeigt während man ein ausgewähltes Element bewegt. Dabei rastet die Komponente an den Stellen ein wo es gut aussehen könnte und zeigt durch eine Linie an, weshalb es dort ein gerastet ist.

In der Abbildung 3.21 sieht man zum Beispiel, dass das ausgewählte Element an dieser Stelle einrastet weil es sich auf der selben Höhe befindet wie das Element das nebenan platziert ist.

Abbildung 3.21: „Snap Line" hilft bei der zentralen Positionierung

Des Weiteren lassen die „Snap Lines" einen erkennen, ob ein Knoten der zwischen 2 anderen Knoten positioniert wird sich mittig befindet oder ob ein Knoten die selbe Höhe oder Breite hat als ein anderer Knoten.

3.7.3.2 Grid

Eine weitere Hilfestellung für ein gutes Layout ist das Grid, zu deutsch Gitter, das man über die Menüleiste einschalten kann. Hat man es eingeschaltet, rasten die Elemente die man aus der Palette zieht an einem der fest vorgegebenen Punkte ein.

3.7.3.3 Orthogonale Kanten

Das letzte Feature das Abhilfe bei der händischen Anordnung der Komponenten schafft sind die orthogonalen Kanten. Hat man diese Funktion über die Menüleiste von yEd aktiviert, dann werden Kanten nur noch mit 90° Winkeln erstellt. Das heißt, dass es keine schiefen Linien mehr gibt, sondern der Editor von sich aus die Linien horizontal bzw. vertikal zeichnet bis die gewünschten Knoten miteinander verbunden sind, wie in Abbildung 3.22 dargestellt wird.

3.7.4 Layout-Algorithmen

Einer der größten Vorteile die yEd besitzt, sind die Layout-Algorithmen die der Editor einem zur Verfügung stellt. Insgesamt

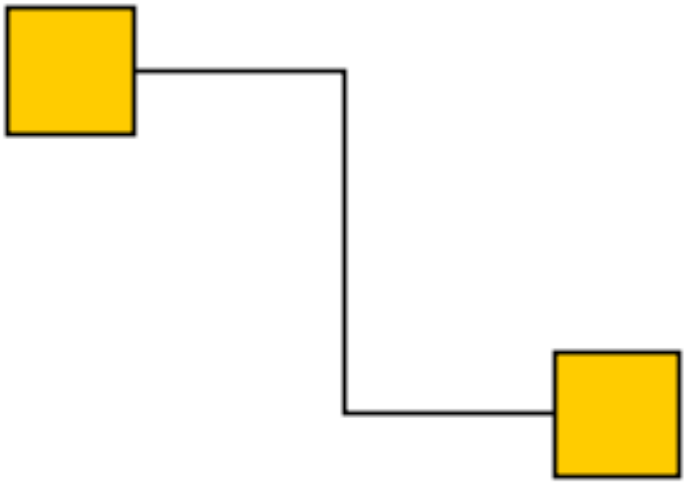

Abbildung 3.22: Beispiel einer orthogonalen Kante in yEd

3.8 GraphML

3.8.1 Ursprung

3.8.2 Verwendung

3.8.3 Darstellungsmöglichkeiten

3.8.3.1 title

3.8.3.2 title

3.8.4 Formatierung mit yEd

3.8.4.1 Automatische Formatierung

3.8.4.2 Vorgefertigte Formatierungsmöglichkeiten

3.9 Graphviz

3.9.1 Allgemeines

3.9.2 Dateiformate

3.9.3 Engines

3.9.4 Vor- und Nachteile von Graphviz

3.9.5 Vergleich von dot und neato

3.10 LibreOffice Draw

3.10.1 Einführung in LibreOffice Draw

Das Grafikprogramm Draw ist Teil des LibreOffice Pakets und wurde von der „The Document Foundation“ entwickelt. Andere Bestandteile des Libre Office Pakets sind Writer,

Calc, Impress, Base und Math. Draw bietet diverse Funktionalitäten um Skizzen, Poster, technische Zeichnungen, Flussdiagramme, etc. zu erstellen. Der Funktionsumfang von Draw gliedert sich in etwa zwischen den Grafikprogrammen Paint.NET und GIMP ein. Im Gegensatz zu Paint stehen mehrere vordefinierte 2D- und 3D-Formen zu Verfügung. Allerdings sind viele grafische Bearbeitungsfunktionen, die GIMP zu Verfügung stellt, nicht enthalten. Draw verwendet für die Speicherung und Darstellung der gezeichneten Objekte ein Koordinatensystem, wobei der Punkt [0;0] sich in der linken oberen Ecke der Seite befindet.

3.10.2 Generierungsvarianten

Um ein Entity Relationship Diagramm in LibreOffice Draw automatisiert generieren zu können, bieten sich zwei Möglichkeiten an, die sich in der Komplexität und der Umsetzung sehr stark unterscheiden. Die erste der beiden Möglichkeiten ist der Zugriff über die, zu Verfügung stehende, API. Die zweite Variante ist die Manipulation des Dateiformats.

3.10.2.1 Generierung über die LibreOffice API

3.10.2.1.1 Allgemeines

Auf der Website der LibreOffice API befinden sich diverse Beispiele um die Funktionsweise und die Verwendung der API zu demonstrieren. Die verschiedenen Beispiele unterscheiden sich in der verwendeten Programmiersprache und dem Zielprogramm. Als Programmiersprachen stehen Java, C++, Python, Basic und das Objektsystem Object Linking and Embedding zur Verfügung. Bei der genaueren Betrachtung zeigt sich, dass das einzige Beispiel, dass Draw als Zielprogramm hat, in Java geschrieben ist.

Um in ein LibreOffice Draw Dokument zeichnen zu können, muss zuerst ein Dokument, eine Instanz der Klasse com.sun.star.frame.Desktop mit einem Kontext der Klasse com.sun.star.uno.XComponentContext und einen Komponentenlader mit der Instanz der Klasse com.sun.star.frame.Desktop angelegt werden. In der folgenden Grafik befindet sich ein Beispiel für jenen Code.

```
com.sun.star.frame.XComponentLoader xCLoader;
com.sun.star.lang.XComponent xComp = null;
try {
    // get the remote office service manager
    com.sun.star.lang.XMultiComponentFactory xMCF = xContext.getServiceManager();

    Object oDesktop = xMCF.createInstanceWithContext("com.sun.star.frame.Desktop", xContext);

    xCLoader = UnoRuntime.queryInterface(com.sun.star.frame.XComponentLoader.class, oDesktop);
    com.sun.star.beans.PropertyValue szEmptyArgs[] = new com.sun.star.beans.PropertyValue[0];
    String strDoc = "private:factory/sdraw";
    xComp = xCLoader.loadComponentFromURL(strDoc, "_blank", 0, szEmptyArgs);

} catch (java.lang.Exception e) {
    System.err.println(" Exception " + e);
    e.printStackTrace(System.err);
}
```

Abbildung 3.23: Beispielcode

Über einige weitere Schritte kommt man so auf ein Objekt der Klasse xShapes, in das man dann die zu zeichnenden Objekte hinzufügt.

```
xDrawDoc = openDraw(xContext);
try {
    System.out.println("getting Drawpage");
    com.sun.star.drawing.XDrawPagesSupplier xDPS = UnoRuntime
            .queryInterface(com.sun.star.drawing.XDrawPagesSupplier.class, xDrawDoc);
    com.sun.star.drawing.XDrawPages xDPn = xDPS.getDrawPages();
    com.sun.star.container.XIndexAccess xDPi = UnoRuntime
            .queryInterface(com.sun.star.container.XIndexAccess.class, xDPn);
    xDrawPage = UnoRuntime.queryInterface(com.sun.star.drawing.XDrawPage.class, xDPi.getByIndex(0));
} catch (java.lang.Exception e) {
    System.err.println("Couldn't create document" + e);
    e.printStackTrace(System.err);
}
com.sun.star.drawing.XShapes xShapes = UnoRuntime.queryInterface(com.sun.star.drawing.XShapes.class, xDrawPage)
```

Abbildung 3.24: Beispielcode2

Mit dem in Abbildung 1 und 2 gezeigten Programmcode, öffnet sich ein LibreOffice Draw Dokument ohne Inhalt.

3.10.2.1.2 Darstellung der grafischen Elemente

Die Darstellung der verschiedenen grafischen Elemente folgt immer den gleichen Schema. Zuerst wird ein Objekt, welches die derzeitige Seite des Dokuments beinhaltet, angelegt. Dieses wird dann in ein Objekt der Klasse xDrawPage umgewandelt. Parallel dazu benötigt man ein Objekt der Klasse XShape, welches eine Vorlage beinhaltet. Die Vorlage bestimmt über die Form des grafischen Elements.

Für die Erstellung der Komponenten eines ERDs sind folgende Klassen notwendig:

- RectangleShape für die Erstellung von Rechtecken um Entity-Typen darzustellen.
- EllipseShape für die Erstellung von Ellipsen um Attribute darzustellen.
- PolyPolygonShape für die Erstellung von Diamanten und Dreiecken um Beziehungen und Super-Sub-Typen darzustellen. Die Klasse kann Körper mit n Seiten darstellen.
- ConnectorShape für die Erstellung von Linien, die Entity-Typen mit Attributen und Entity-Typen mit Beziehungen verbindet. Im Gegensatz zu der Klasse LineShape, die es nur ermöglicht die Endpunkte der Linie an eine bestimmte Koordinate zu binden, besteht die Möglichkeit die Endpunkte der Linie mit einem Objekt einer anderen Klasse zu binden. Dies unterbindet ein aufwendiges Verschieben der Linie, falls ein Entity-Typ oder eine Beziehung verschoben wird.
- XText für die Erstellung von Textfeldern um die Namen der Entity-Typen und der Attribute anzuzeigen. Außerdem benötigt man für die min-max Notation ebenfalls Textfelder.

In allen oben gelisteten Klassen besteht die Möglichkeit dem Objekt eine Position zuzuweisen. Wird keine Position explizit angegeben, so werden die Koordinaten [0;0] verwendet, wobei sich die Koordinaten auf den linken oberen Punkt des Objekts bezieht. Die Funktion setSize der Klasse xDrawShape setzt die Breite und Höhe des Körpers fest. Standardmäßig

werden die Objekte mit einem schwarzen Rand und weißer Füllung gezeichnet. Mit der Funktion setPropertyValue der Klasse XPropertySet kann man die Objekte in einer beliebigen Farbe darstellen. Die Farbwerte werden im Dezimalformat angegeben.

Die folgende Abbildung zeigt den Programmcode um ein Rechteck ohne Farbe zu erstellen:

```
Object drawPages = xDrawPagesSupplier.getDrawPages();
XIndexAccess xIndexedDrawPages = (XIndexAccess) UnoRuntime.queryInterface(XIndexAccess.class, drawPages);
Object drawPage = xIndexedDrawPages.getByIndex(0);
XMultiServiceFactory xDrawFactory = (XMultiServiceFactory) UnoRuntime
        .queryInterface(XMultiServiceFactory.class, xComponent);
Object drawShape = xDrawFactory.createInstance("com.sun.star.drawing.RectangleShape");
XDrawPage xDrawPage = (XDrawPage) UnoRuntime.queryInterface(XDrawPage.class, drawPage);
xDrawShape = UnoRuntime.queryInterface(XShape.class, drawShape);
xDrawShape.setSize(new Size(width, height));
xDrawShape.setPosition(new Point(x, y));
xDrawPage.add(xDrawShape);
XText xShapeText = UnoRuntime.queryInterface(XText.class, drawShape);
XPropertySet xShapeProps = UnoRuntime.queryInterface(XPropertySet.class, drawShape);
xShapeProps.setPropertyValue("FillColor", Integer.valueOf(col));
com.sun.star.text.XTextCursor xTCursor = xShapeText.createTextCursor();
com.sun.star.beans.XPropertySet xTCPS = UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class,
        xTCursor);
xTCPS.setPropertyValue("CharHeight", 6.0f);
xShapeText.insertString(xTCursor, text, false);
com.sun.star.beans.XPropertySet xText = UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class,
        xShapeProps);
```

Abbildung 3.25: Beispielcode3

3.10.2.2 Generierung über das Dateiformat

3.10.2.2.1 Allgemeines

LibreOffice Draw verwendet das OpenDocument Graphics (Kurzform: ODG) Dateiformat, welches 2006 als internationale Norm ISO/IEC 26300 veröffentlicht wurde. ODG basiert auf einem XML Vokabular, dessen Elemente an den Standard HTML angelehnt sind. Eine OpenDocument-Datei besteht aus einer Sammlung mehrerer XML-Dateien und anderer Objekte wie Bilder oder Thumbnails, die zu einer Datei im ZIP-Format zusammengefasst werden Bei der Entpackung dieser ZIP-Datei sieht man folgende Dateien und Ordner:

```
Datei.odt
|
+-- META-INF
|    |
|    +-- manifest.xml
|
+-- Thumbnails
|    |
|    +-- thumbnail.png
|
+-- Pictures
|    |
|    +-- picture.png
|
```

```
+-- mimetype
+-- content.xml
+-- styles.xml
+-- meta.xml
+-- settings.xml
```

Inhalt dieser Dateien:

- manifest.xml: In der Manifest-Datei befinden sich eine Übersicht aller Dateien mit deren Pfaden und erlaubten Datentypen.
- thumbnail.png: Die Thumbnail-Datei zeigt eine Miniaturansicht der ersten Seite des Dokuments.
- Im Pictures Ornder befinden sich alle eingebundenen Bilder des Dokuments. Falls keine Bilder eingebunden sind, existiert der Ornder nicht.
- mimetype: Die Mimetype Datei ist eine Textdatei, dessen einziger Inhalt eine Zeile mit dem Typ der LibreOffice-Datei ist. Im Beispiel einer LibreOffice Draw Datei steht folgender Inhalt in der Mimetype-Datei:

  ```
  application/vnd.oasis.opendocument.graphics
  ```

- settings.xml: In der Settings-Datei befinden sich sämtliche Einstellungsmöglichkeiten, die LibreOffice Draw anbietet. Darunter fallen zum Beispiel die Maßeinheit, Name und Setup der Drucker, die Skalierung und die Sichtbarkeit der einzelnen Ebenen.
- content.xml: In der Content-Datei befindet sich der eigentliche Inhalt des Dokuments. Ein leeres Draw Dokument ist wie folgt aufgebaut:

  ```
  Element: office:document-content
  |
  +-- Attribut: xmlns:--
  +-- Element: office:scripts
  +-- Element: office:font-face-decls
  |
  +-- Element: style:font-face
  |
  +-- Attribut: style:name
  +-- Attribut: style:font-family
  +-- Attribut: style:font-family-generic
  +-- Attribut: style:font-pitch
  +-- Element: office:automatic-styles
  |
  +-- Element: style:style
  |
  +-- Attribut: style:name
  +-- Attribut: style:family
  +-- Element: office:body
  |
  +-- Element: office:drawing
  |
  +-- Element: draw:page
  |
  ```

```
+-- Attribut: draw:name
+-- Attribut: draw:style-name
+-- Attribut: draw:master-page-name
```

Sämtlicher manuell eingefügter Inhalt wird als Kindelement von dem Element draw:page eingefügt. In dem Element office:document-content befinden sich mehrere Attribute mit diversen Namensräumen, die aus Gründen zur Übersichtlichkeit weggelassen wurden.

3.11 Git

Bei einem Projekt wie diesem ist es von Vorteil jegliche Art von Versionsverwaltung zu verwenden. Außer Git hätten sich noch andere Versionsverwaltungssysteme angeboten wie z.B.:

- Mercurial
- Darcs
- Fossil
- Bazaarn

Jedoch wurde für dieses Projekt geziehlt Git verwendet. Git ist unter den Versionverwaltungssystemen das gängigste und deswegen existiert dafür auch der meiste Support. Außerdem verwendet Trac, die Projektorganisations-Web-App die in diesem Projekt verwendet wurde, Git deswegen waren die anderen Möglichkeiten nicht mit Git gleichwertig.

Durch geziehltes anlegen von Branches wurde es ermöglicht jedem Entwickler eine vollkommen von den Anderen unabhängige Entwicklungsumgebung zu bieten. Diese sogennanten Feature-Branches ermöglichten es dass, unabhägig vom derzeitigen Entwicklungsstands des Projekts, es zu jederzeit eine funktionierende Version der Software am Master-Branch vorlag.

Die Daten lagen stehts auf dem Server, den der Auftraggeber bereitgestellt hat, vor und ermöglichte dadurch eine einfachen Zugriff. Mit einem unabhängigen Server ist es den Entwicklern leicht gefallen von zu Hause und in der Schule immer am neusten Stand zu sein.

3.12 Trac

Trac ist eine Software die bei der Entwicklung und Organisation des Projekts eine große Hilfe ist. Dadurch wird ein Interface für Versionsverwaltung geboten, sowie ein für Scrum ideales Ticketsystem. Außerdem bietet es einen Aktivitätsüberblick und einen Überblick über den aktuellen Stand des Projekts in Form von einer Roadmap.
Durch das Ticketsystem kann man mit Leichtigkeit die verschiedenen Aufgaben verteilen und feststellen ob sie fertiggestellt wurden.

Außerdem erlaubt Trac WikiFormatting das die meisten von Wikipedia kennen. Mithilfe diesem Merkmals kann man eine angenehme und leicht leserliche Struktur erschaffen.
Davon war das erstellen von Links auf interne Ressourcen sowie auf externe Ressourcen eine der meist verwendeten Funktionen.

#6 new User Story

Opened 5 months ago

UD002 Als Anwender möchte ich dass mein generiertes ERD mit einem gut lesbaren Layout generiert wird

Reported by:	Nicolas Homolka	Owned by:	
Priority:	1	Milestone:	Product Backlog DA
Component:	Diplomarbeit	Version:	
Keywords:		Cc:	

Description

Das generierte ERD soll in einem gut lesbaren Layout ausgegeben werden

↳ Reply

Abbildung 3.26: Ticket im Trac

Links	`http://trac.edgewall.org`	http://trac.edgewall.org
	`WikiFormatting (CamelCase)`	WikiFormatting (CamelCase)
TracLinks	`wiki:WikiFormatting`, `wiki:"WikiFormatting"`	wiki:WikiFormatting, wiki:"WikiFormatting"
	`#1 (ticket)`, `[1] (changeset)`, `{1} (report)`	~~#1~~ (ticket), [1] (changeset), {1} (report)
	`ticket:1, ticket:1#comment:1, comment:1:ticket:1`	~~ticket:1~~, ~~ticket:1#comment:1~~, ~~comment:1:ticket:1~~
	`Ticket [ticket:1], [ticket:1 ticket one]`	Ticket ~~1~~, ~~ticket one~~
	`Ticket [[ticket:1]], [[ticket:1|ticket one]]`	Ticket ~~1~~, ~~ticket one~~

Abbildung 3.27: Dokumentation für Links im Trac

Author:

Kapitel 4

Ergebnis

4.1 CLI

4.1.1 Warum eine CLI verwenden?

4.1.2 Vorteile

4.1.3 Nachteile

4.1.4 Andere Interagierungsmöglichkeiten

4.2 ERD

4.2.1 Entstehung

4.2.2 Aufbau eines ERD's

4.2.3 Entity Typen

4.2.4 Beziehungen zwischen Entity Typen