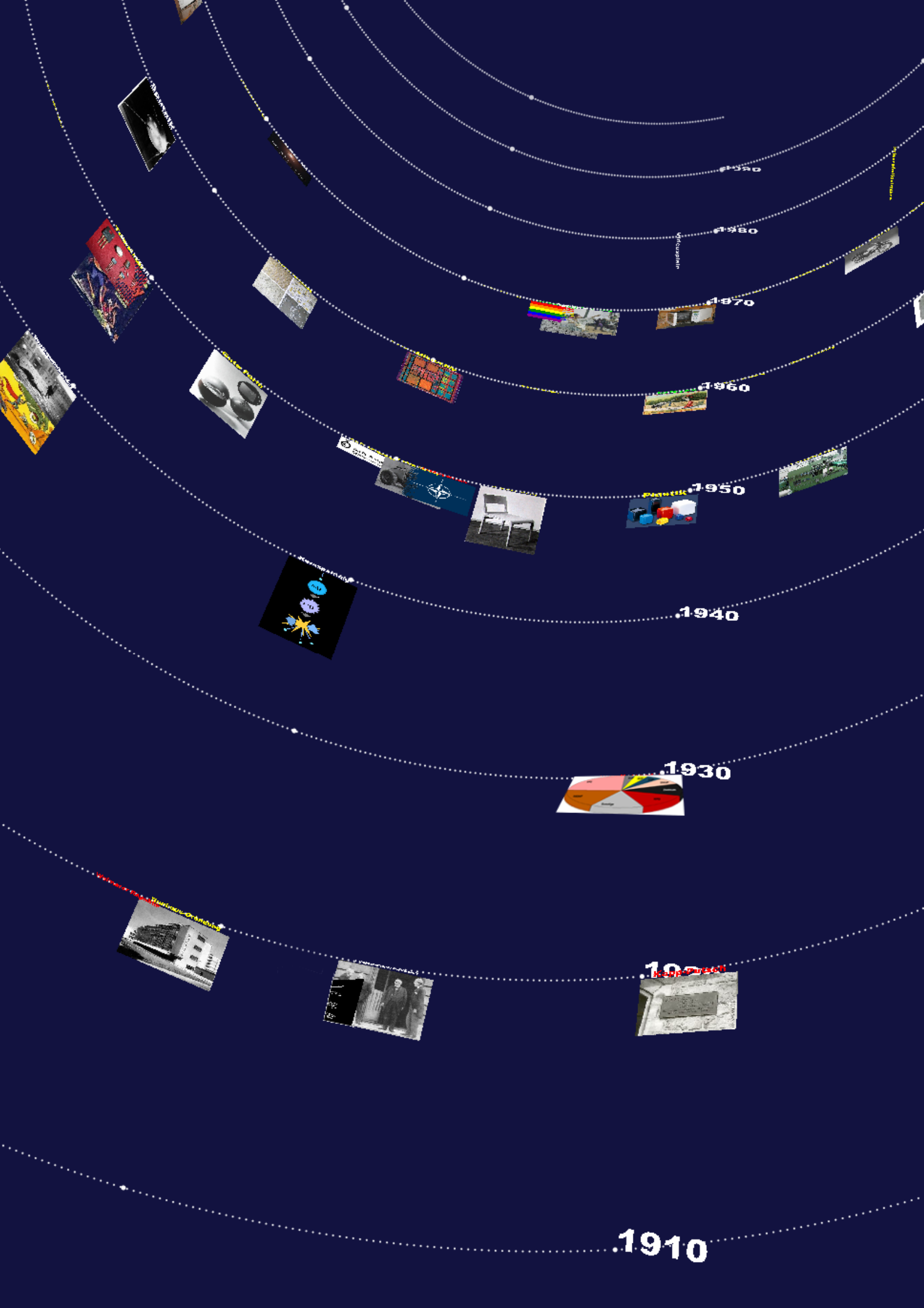
Großer Beleg

Visualisierung großer Datenmengen im Raum

Jan Polowinski
geboren am 3.3.1981 in Gelsenkirchen

Technische Universität Dresden
Fakultät Informatik
Institut für Software- und Multimediatechnik

Betreuung durch Prof. Dr.-Ing. habil. Rainer Groh
und Prof. Dr. rer. nat. habil. Uwe Aßmann
abgeschlossen am 12. Juni 2006


1980
1970
1960
Plastik
1950
1940
1930
1910

Inhalt

Abstract

Large, strongly connected amounts of data, as collected in knowledge bases or those occurring when describing software, are often read slowly and with difficulty by humans when they are represented as spreadsheets or text. Graphical representations can help people to understand facts more intuitively and offer a quick overview. The electronic representation offers means that are beyond the possibilities of print such as unlimited zoom and hyperlinks.
This paper addresses a framework for visualizing connected information in 3D-space taking into account the techniques of media design to build visualization structures and map information to graphical properties.

Keywords

data visualisation, visualization of chronological data, visualization of relations, ontology browser, graphical vocabulary

Meta-Informationen zum Dokument

Begriffe von Konzepten, die durchgängig verwendet werden, sowie Namen von Konstrukten im Programmcode oder den Ontologien, die im Text Erwähnung finden, sind bei ihrer ersten Nennung und bei besonderer Betonung des Eigennamens in der Schriftart Arial gedruckt. Von einer generellen Kennzeichnung habe ich abgesehen, weil stellenweise zu viele Begriffe in Frage kamen, so dass der Lesefluss gestört worden wäre.
Im Kapitel „Framework und Gestaltung des Mediums“ sind Konzepte, die in der Beispielimplementierung noch nicht umgesetzt sind, durch ein Plus-Zeichen in eckigen Klammern [+] markiert.

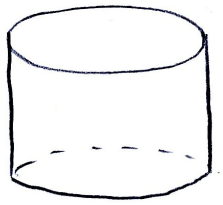
1 Einführung

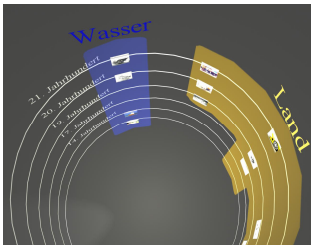
Dieser Abschnitt erläutert die Ideen, die zur Bearbeitung des Beleg-Themas führten, und nennt die Zielsetzung. Abschließend werden alle Fachbereiche genannt, welche die Arbeit berührt. Auf die einzelnen Bereiche wird im Verlauf dieser Ausarbeitung näher eingegangen.

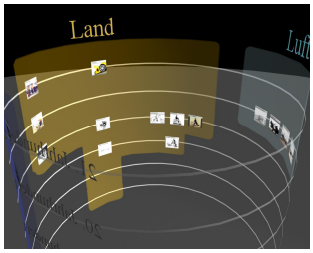
1.1 Zusammenfassung des Gestaltungsentwurfs

Diese Belegarbeit führt einen Gestaltungsentwurf für ein Programm zur Visualisierung von (vorwiegend zeitlichen) Daten im 3D-Raum fort. [BRADE POLOWINSKI 2004]. In dem Komplexpraktikum entstanden drei verschiedene Modelle zur Anordnung zeitlicher Daten im 3D-Raum:

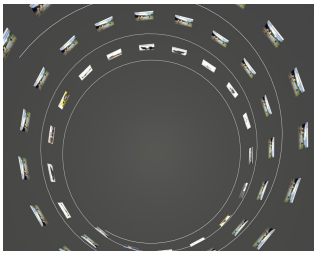
a) Ein Zylinder – bei dem die Position in Y-Richtung den Zeitpunkt eines Ereignisses und die Position auf der Oberfläche eine Einteilung in eine bestimmte Kategorie bedeutet.

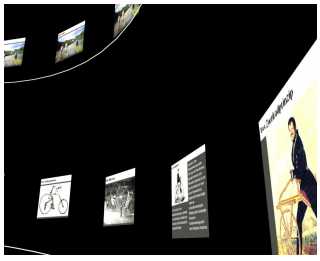




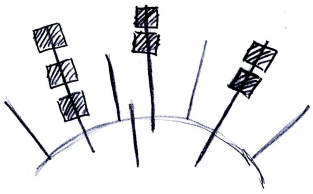


b) Eine Wendel – bei der eine Drehung einen zeitlichen Zyklus (Jahre, Jahrzehnte …) bedeutet.





c) Ein geographisches Modell in Globusform – bei dem der geographische Ort eines Ereignisses im Vordergrund steht und Zeitstrahlen vom Mittelpunkt nach außen führen, an denen die Darstellungen der Ereignisse positioniert sind.



Motiviert waren diese Überlegungen durch den Wunsch, Zeitgeschichte zugleich übersichtlich, aber auch im Detail verständlich zu machen. Die Dimension von zeitlichen Abständen, die Tiefenzeit[1] [GOULD 1987], sollte genauso sichtbar sein wie die genaue relative wie absolute Lage der Ereignisse. Der menschlichen Wahrnehmung der Zeit in astronomischen Zyklen sollte Rechnung getragen werden. Nicht zuletzt sollten kausale Beziehungen als oft relevantere Information gut ablesbar sein.

1.2 Ziel des Belegs

Das primäre Ziel war der Entwurf eines Frameworks zur Datenvisualisierung, welches das Experimentieren mit gestalterischen Parametern ermöglicht. Daneben sollte eine beispielhafte Umsetzung des Helix-Modells aus dem gestalterischen Entwurf mit Hilfe dieses Frameworks vorgenommen werden, um damit Beispieldaten zu visualisieren. Weiterhin sollten folgende Aspekte berücksichtigt werden:

- Schaffen von Möglichkeiten zum Einsatz der Erkenntnisse der Mediengestaltung.
- Betonung der Beziehung zwischen Dingen als wichtige Information und Versuch der geeigneten Darstellung.
- Schaffen einer praktischen Anwendung von Ontologien im Bereich der Datenvisualisierung. Hierbei sollte der Schwerpunkt jedoch auf intuitive Verständlichkeit statt auf Vollständigkeit wie bei Ontologiebrowsern[2] gesetzt werden.
- Verfolgen eines möglichst allgemeinen Ansatzes, im Hinblick auf Visualisierung von anderen Daten wie auch von Software.
- Berücksichtigung der Ideen der Facettenklassifizierung [PRISS 2000].

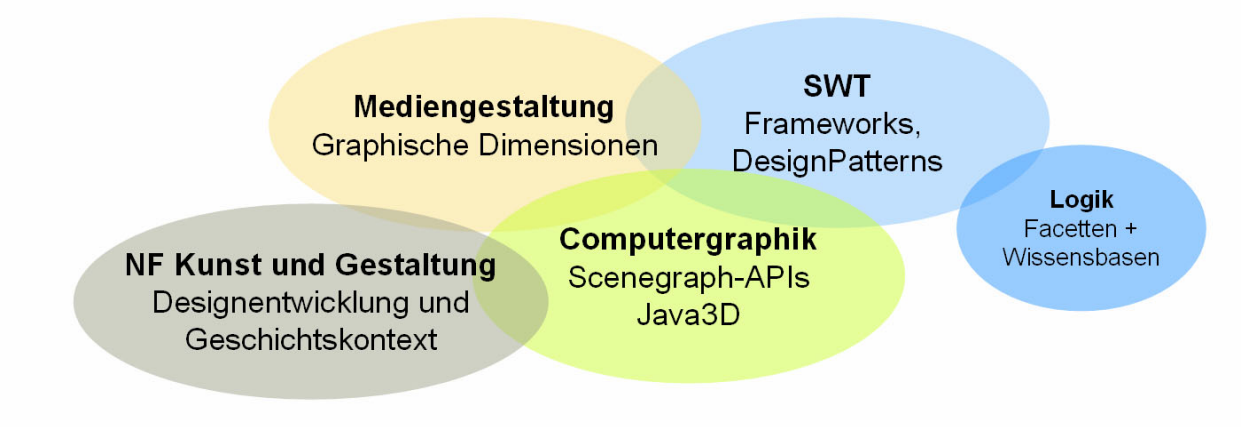
1.3 Interdisziplinäres Projekt

Das Projekt berührt Bereiche der Mediengestaltung, aber auch in hohem Maße der Softwaretechnik, Computergraphik und in geringerem Maße der Logik.
Der Einsatz von Metaphern, das Abstimmen gestalterischer Parameter der Modelle, Farbschemata und die Typographie betreffen die Mediengestaltung.
Die Softwaretechnik liefert die nötigen Verfahren zur Erstellung eines flexiblen Frameworks, wie es zum Experimentieren mit den gestalterischen Parametern nötig ist. Erweiterbarkeit und Wiederverwendung sind entscheidend, um die Flexibilität dauerhaft zu ermöglichen.
Die verwendete Scenegraph-API Java3D [JAVA3D], auf die das Framework aufbaut, betrifft den Bereich der Computergraphik. Obwohl der Abstraktionsgrad von Java3D sehr hoch ist, kamen einige Grundlagen der Computergraphik während der Arbeit am Beleg zum Tragen, wie z.B. die Transformation von Matrizen oder Techniken wie Level of Detail und Texture-Mapping.

---

[1] Als Tiefenzeit bezeichnet Gould die dem Menschen erst durch geologische Forschung bewusst gewordenen Ausmaße der Zeit, von denen der Mensch nur einen geringen Bruchteil existiert [GOULD 1987].
[2] Ontologiebrowser dienen der graphischen Darstellung von Ontologien

Die Daten wurden in Form von Ontologien gespeichert, um stark verknüpfte Daten besonders günstig ablegen zu können. So berührt die Arbeit den Bereich der Wissensbasen und deshalb war eine intensive Auseinandersetzung mit Upper-Level-Ontologien sowie Ontologie-Konstruktion nötig. In diesem Zusammenhang fand auch die Facettentheorie Beachtung [PRISS 2000]..
Als Beispielinhalte für die Visualisierung wurden im Rahmen der Kooperation mit einer Arbeit im Fach „Kunst und Gestaltung“ designgeschichtliche Daten vorgesehen. (Siehe Seite 13, Konkrete Beispielinhalte).



2 Vorgehen

Dieses Kapitel beschreibt die Vorgehensweise bei der Bearbeitung des Projekts und geht auf die Aufteilung des Arbeitsaufwands zwischen den beiden in Kooperation durchgeführten Arbeiten „Großer Beleg“ und einer im Fach „Kunst und Gestaltung“ angefertigten Abschlussarbeit ein.

2.1 Ablauf

Im Folgenden eine Übersicht über die Teilarbeitsschritte im Verlauf der Arbeit an dem Projekt:

Planungsphase

- Planung: Mindmaps und Skizzen
- Recherche zur Datenvisualisierung und dem Belegthema ähnlicher Projekte
- Recherche Java 3D-Grafik APIs
- Tests und Einarbeitung Java3D [JAVA3D]

Analyse

- Recherche und Sammeln von Bildmaterial zum Thema „Designgeschichte Deutschlands im 19. und 20. Jahrhundert“
- Recherche der Kontextereignisse
- Analyse der Domain Designgeschichte
- Beschäftigung mit der Facettenklassifizierung

Prototyp

- Recherche Upper-Level-Ontologien, Einarbeitung Protégé + Jena-API [PROTEGE, JENA]
- Erstellen einer Ontologie
- Einpflegen der Inhalte als Testdaten
- Prototyperstellung

Entwurf und Implementierung

- Entwurf in UML mit dem Programm JUDE-Community™ [JUDE]
- Erstellen von Diagrammen des Szenengraphen
- Umfassende Überarbeitung des Prototypen durch Refactoring
- Experimentieren mit dem Framework unter Berücksichtigung mediengestalterischer Grundsätze
- Vervollständigen der Inhalte, Übersetzung ins Deutsche

Dokumentation

- Reinschrift von Skizzen und Überlegungen, Aktualisieren der Diagramme
- Tests mit der Powerwall[3] zur stereoskopischen Darstellung

Ein wesentlicher Teil der Arbeitszeit ist in die Erstellung des Datenmodells des Designthemas, welches sich in der Ontologie wieder findet, und in die Auseinandersetzung mit Prinzipien und Techniken der Ontologiekonstruktion geflossen. Weitere zeitintensive Schwerpunkte waren die Recherche der Designereignisse und die Einarbeitung in Java3D sowie die Ontologie-API Jena.

[3] Eine Powerwall ist ein Aufbau zur 3D- Präsentation. Zwei Bilder werden auf eine Leinwand projiziert und können dann mit einer Spezialbrille dreidimensional wahrgenommen werden. An der Fakultät Maschinenbau steht an der TU-Dresden eine solche Einrichtung zu Verfügung.

2.2 Konkrete Beispielinhalte

Um konkrete Daten für die Visualisierung zu erhalten, wurde der Beleg parallel zu einer Abschlussarbeit im Fach „Kunst und Gestaltung“ bearbeitet. Das dort benötigte Programm zur Vermittlung von Designgeschichte sollte der Große Beleg liefern, umgekehrt sollten die Inhalte im Großen Beleg als Beispielinhalte dienen.
Visualisiert werden sollte das Thema „Designgeschichte des 19. und 20. Jahrhunderts in Deutschland“ mit Schwerpunkt bei den Unterschieden zwischen Ost- und Westdeutschland. Um die Fülle der Informationen etwas zu begrenzen, wurde der Zeitrahmen auf das 20. Jahrhundert begrenzt, wobei zudem die Jahre nach dem 2. Weltkrieg im Mittelpunkt stehen. Um die designgeschichtlichen Inhalte besser in den Zusammenhang einordnen zu können, wurden außerdem für jedes Jahrzehnt des 20. Jahrhunderts ca. fünf Ereignisse aus den Bereichen Politik, Kultur, Wirtschaft und Technologien herausgesucht, die prägend für ihre Zeit waren und als Kontextereignisse dienen.
Durch die Arbeit am konkreten Beispiel haben sich ständig neue Probleme ergeben, für die das Programm eine Lösung bieten musste. Die Kombination von Inhaltsaufarbeitung und Programmgestaltung hat somit neue Ideen hervorgebracht und einen entscheidenden Mehrwert bedeutet (siehe auch Anhang B).

2.3 Beispielimplementierung

Bei der Erstellung des Frameworks und der Beispielimplementierung des Helix-Modells habe ich mich für eine prototypenbasierte Vorgehensweise entschieden, um sicher zu stellen, dass am Ende der Arbeit ein funktionstüchtiges Tool für die Präsentation des Nebenfaches zur Verfügung steht. So wurde in den verschiedenen Problemgebieten jeweils eine beispielhafte Lösung verfolgt, also vertikal durch alle Bereiche vorgegangen, ohne eine Vollständigkeit anzustreben. Ideen für Verbesserungen sind über „// TODO“ Tags an den jeweiligen Stellen im Quellcode vermerkt und können als Ansatzpunkt für spätere Verbesserungen genutzt werden.
Nur durch die Zusammenlegung der Arbeiten konnte der Aufwand zur Implementierung des Frameworks und der Helix als Beispielanwendung in der gegebenen Zeit bewältigt werden. Zudem konnte die Zeit eingespart werden, die zur Eingabe von sinnvollen Testdaten nötig gewesen wäre.

3 Datenmodell

Die anzuzeigenden Informationen sind in einer Wissensbasis gespeichert. Das können Datenbanken, XML-Dateien oder Ontologien sein. In der Beispielimplementierung wurde eine Ontologie verwendet, da sie die vernetzten Informationen sehr gut repräsentiert. Sie stellt das Datenmodell der Anwendung dar und liefert ebenfalls die Daten selbst.
Das folgende Kapitel gibt eine kurze Einführung zu Ontologien und Begriffen der Wissensrepräsentation, die im Zusammenhang mit diesen verwendet werden. Anschließend wird der Konstruktionsprozess der Design-Ontologien beschrieben und auf dabei entstandene Schwierigkeiten eingegangen. Zuletzt wird ein Filtermechanismus für die zur Visualisierung bestimmten Objekte und eine graphische Oberfläche für diesen Filtermechanismus entworfen.

3.1 Ontologien

„Unter einer Ontologie versteht man in der Informatik im Bereich der Wissensrepräsentation ein formal definiertes System von Konzepten und Relationen. Die bekannteste Definition lautet "Spezifikation einer Konzeptualisierung". Zusätzlich enthalten Ontologien Inferenz- und Integritätsregeln. „[4]

Im Unterschied zu einer Abspeicherung in einer Datenbank bieten Ontologien die Möglichkeit, über einen angeschlossenen Reasoner[5] Schlussfolgerungen in das Ergebnis einer Abfrage mit einfließen zu lassen. So schließt die Suche nach Ereignissen automatisch die Menge der Erfindungen mit ein, vorausgesetzt, es wurde definiert, dass eine Erfindung ein Ereignis sei.
Solche Definitionen können und sollten von bestehenden Ontologien übernommen werden. Man bezeichnet solch allgemeine Ontologien als Upper-Level-Ontologien (siehe unten).

3.2 Ontologie Konstruktion

Sprache

Bei der Konstruktion der Ontologie wurde als zugrunde liegende Sprache die Web Ontology Language (OWL) verwendet. Die Sprache wird vom W3C[6] als Sprache für das Semantic Web[7] empfohlen, basiert auf dem Resource Description Framework Schema (RDFS)[8] und stellt einen Kompromiss zwischen Mächtigkeit und effizienter Berechenbarkeit dar. OWL erweitert RDFS, indem es z.B. die Definition transitiver und inverser Relationen bereitstellt und logische Operationen ermöglicht. Viele verwendete Konstrukte, wie rdfs:label, das die Mehrsprachigkeit unterstützt, werden jedoch bereits von RDFS geboten. Weiterhin gibt es gute Werkzeuge für das Editieren von OWL sowie mit Jena[9] eine ausgereifte API in Java zur Abfrage der Informationen.
OWL hat drei in ihrer Ausdrucksmächtigkeit unterschiedlich starke Subsprachen: OWL LITE, OWL DL und OWL FULL, von denen OWL FULL die mächtigste ist.

---

[4] Quelle: Wikipedia
[5] Ein Reasoner ist ein Programm, welches die ihm übermittelten Regeln auswertet und logische Schlüsse daraus zieht
[6] W3C: WorlWideWeb Consortium http://www.w3.org/
[7] Das Semantic Web bietet ein gemeinschaftliches Rahmenwerk, das das Teilen und den anwendungsübergreifenden Gebrauch von Daten ermöglicht. http://www.semanticweb.org/
[8] RDFS: http://www.w3.org/TR/rdf-schema/ und RDF: http://www.w3.org/RDF/
[9] Jena: http://jena.sourceforge.net/ siehe auch Seite 44, Technologien und Werkzeuge

Kurze Übersicht der im Folgenden verwendeten OWL- und RDFS-Konzepte

Eine genaue Beschreibung von OWL findet sich unter http://www.w3.org/TR/owl-ref/

- owl:Class — Klassen
- owl:DataProperty — eine Relation mit einem primitiven Datentyp als Range
- owl:ObjectProperty — eine Relation mit einer oder mehreren Klassen als Range
- rdfs:range — Zielbereich der Relationen.
- rdfs:domain — Definitionsbereich einer Relation

Wiederverwendung

Beim Konstruktionsprozess wurde die der Wiederverwendung dienliche Auftrennung in mehrere Ontologien mit eingegrenztem Sachgebiet eingehalten und außerdem auf eine Trennung von Instanzen und Klassen in unterschiedliche Dateien geachtet. Fakten können so ausgewechselt werden, ohne die Beschreibung der Ontologiestruktur zu berühren.

Upper-Level-Ontologien

Upper-Level-Ontologien definieren Begriffe, die in vielen verschiedenen Domainen benötigt werden. Dabei werden aber nur die der Domain übergeordneten Begriffe definiert und die domainspezifischen bewusst ausgespart. Auf diese Art und Weise gibt es eine gemeinsame Basis für die übergreifende Verwendung verschiedener Ontologien. Der Wert einer Menge von Ontologien steigt, wenn viele Konstrukteure auf eine solche gemeinsame Basis zugreifen.
So wurde auch für die Design-Geschichte-Ontologie das Verwenden einer Upper-Level-Ontologie angestrebt, um eine allgemeine Basis für die Erstellung von darstellbaren Inhalten zu erhalten.
Als gängige Upper-Level-Ontologien haben sich unter anderem folgende Projekte herausgebildet:

- DOLCE: Die Sprache ist KIF[10]. Sie wurde mit dem Ziel entwickelt, eine Standard-Ontologie zu bieten. Eine OWL-Variante ist vorhanden.[11]
- SUMO: Die Sprache der SUMO ist SUO-KIF. Sie ist eine ebenfalls weit verbreitete, umfangreiche Upper-Level-Ontologie (siehe unten), die mit DOLCE in Konkurrenz steht.
- OpenCyc: OpenCyc ist eine Open-Source-Variante von Cyc, einer seit 1984 betriebenen Wissensdatenbank des Alltagswissens[12] und verwendet die Sprache CycL. Eine OWL-Variante ist nicht direkt verfügbar.

Darüber hinaus gibt es Domain-Ontologien, die sich mit bestimmten Themen wie etwa Geographie oder Transportwesen beschäftigen. Die Wahl für das Projekt fiel auf die SUMO, da sie einerseits in OWL erhältlich ist und anderseits einige Konzepte mehr als die DOLCE besitzt, die für das Thema Design interessant sind.

[10] KIF: Knowledge Interchange Format: http://logic.stanford.edu/kif/kif.html
[11] DOLCE: http://dolce.semanticweb.org/
[12] OpenCyc: http://www.opencyc.org

SUMO

Die **S**uggested **U**pper **M**erged **O**ntology von der SUO working group ist eine Upper-Level-Ontologie, die in SUO-KIF[13], einer Variante der Sprache KIF, einer deklarativen Logik-Sprache, entwickelt wird. Die erste Veröffentlichung fand im Jahr 2000 statt. Auf der Website finden sich Versionen in OWL und auch eine Protégé –Version, die sich allerdings beide nicht problemlos verwenden ließen. Vom selben Entwickler gibt es die MILO, eine Midlevel-Ontology, die die Lücke von der sehr allgemeinen SUMO zu Domain-Ontologien schließen soll. Diese liegt beim Entwickler leider nur im KIF Format vor und konnte so nicht mit angemessenem Aufwand verwendet werden. Auch an anderer Stelle gefundene Übersetzungen im OWL Format ließen sich nicht erfolgreich mit dem Editor Protégé verwenden, da sie den Konventionen für OWL-FULL und somit auch OWL-DL nicht vollständig entsprechen.
Da aber die Verwendung eines soliden Editors Voraussetzung für die Eingabe der Menge an Daten war, kommt die etwas reduzierte Version der SUMO in Kombination mit der neu geschriebenen Midlevel Ontology Middle.owl zum Einsatz. Es sollte jedoch möglich sein, die Design-Ontologie mit geringen Änderungen auf die originale SUMO zu begründen.

Die folgende Abbildung zeigt an einen Ausschnitt aus dem SUMO Unterklassenbaum die Klasse, die für die Designgegenstände als Basis genommen wurde: Artifact.

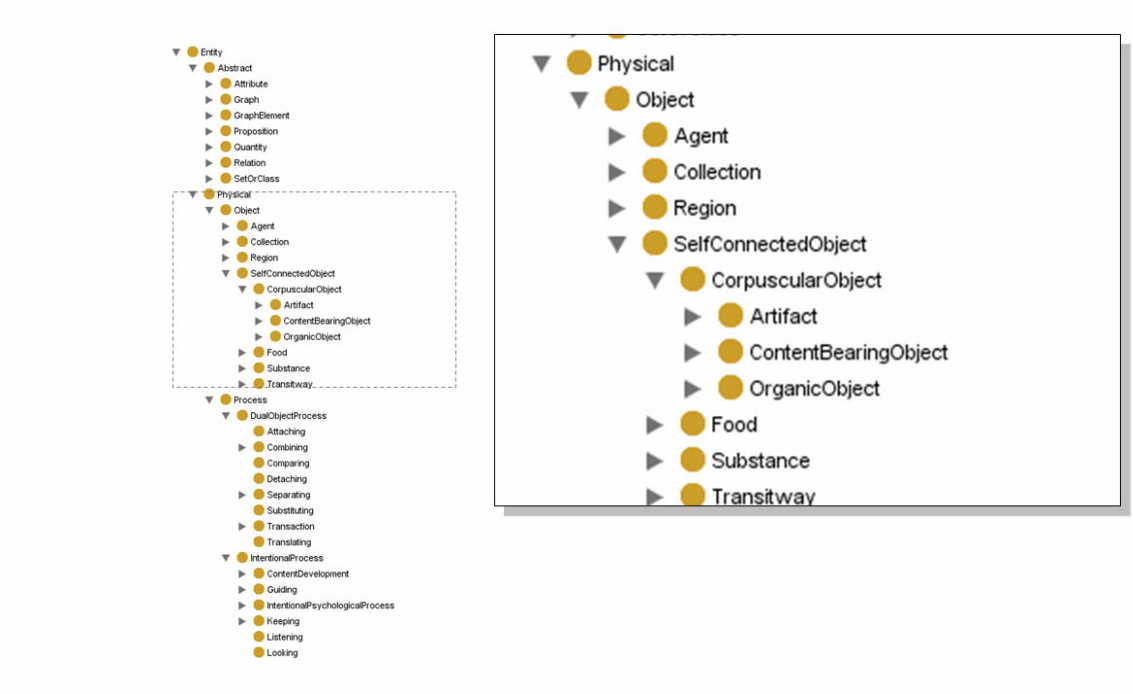


Abbildung 1 - Sumo Unterklassenbaum

[13] SUO-KIF: http://suo.ieee.org/SUO/KIF/suo-kif.html

3.3 Analyse der Domain Design

In einem ersten Schritt der Auseinandersetzung mit der Domain Design, insbesondere der Geschichte des Designs, wurden Texte aus Designbüchern analysiert und zerlegt. Weitere Begriffe, die im Zusammenhang mit der Domain Design wichtig erschienen, wurden aus den Ergebnissen eines Brainstormings ergänzt. Die folgende ungeordnete Liste ist dabei entstanden:

- Ereignis
- Zeitperiode
- Etappe/Epoche
- Jahr
- Zeitpunkt
- Gegenstand
- Bezeichnung
- Designstilrichtung
- Stilrichtung
- Trend, Mode
- Gruppe
- Künstlerbund
- Künstler / Erfinder
- Mensch A kennt Mensch B, B kennt A, A und B kennen sich
- Hersteller (Firma)
- Staat
- Stadt (Ost-West-Zugehörigkeit über Tabelle, oder explizit?)
- Ort
- Ausstellung
- Messe
- Preis (Auszeichnung)
- Material {Holz, Bugholz, Kunststoff, Plastik, Gummi, Metall, Papier}
- Materialvielfalt
- Vorbild in der Natur
- Rundheit
- Beliebtheit / Aktualität heute (Zeitlosigkeit)
- Verbreitung / Durchsetzung
- Maße, Gewicht
- Farbe {rot, grün, blau, gelb, grau, schwarz, weiß}
- Farbigkeit {bunt, einfarbig}
- Textur {gemustert}
- Herstellungsverfahren
- Zweck des Produkts {schneiden, backen, Sitz-Gelegenheit bieten, schreiben}Unterschied zu Einsatzort/-gerät!
- Ähnliche Produkte → Ähnlichkeit → Aussehen/Funktion
- Museum
- Foto
- Quellennachweis
- Datum
- Boolean: public domain
- Variante
- geographische Position
- Bedeutung
  - nach eigenem Ermessen
  - nach allgemeinem Ermessen (z.B. ausgelotet durch Wikipedia)
  - nach Anzahl der Ereignisse, die davon abhängen
  - nach Anzahl der Ereignisse, die es umschließt/verallgemeinert (nur bei Zeitspannen)
- Startzeit
- Dauer, Endzeit, Zeitdauer, „Zeitpunkt“
- beteiligte Personen
- Ursachen (beeinflussende Dinge)
- Wirkungen (Dinge, die beeinflusst werden)
- Entwicklungsrichtungen, die das Ereignis prägten
- Entwicklungsrichtungen, die das Ereignis prägte
- Einsatzgebiete {Küche, Medizin, Verkehr, Wohnen, Waffen, Industrie, Kleidung}
- Einsatzgerät: {Stuhl, Tisch, Bett, Mixer, Lampe, Auto, Flugzeug, Eisenbahn, Tapete, Trinkgefäß, Maschine, Automat, Telekommunikationsgerät, Telefon, Werbung, Poster, Sportgerät, Kleidung} (Was geschieht bei Verschmelzung von Geräten und wenn neue entstehen?)
- Kontextereignisse, die die Designentwicklung beeinflussten (Fertigungsverfahren, Energie, Transport, Kommunikation, Architektur, Wissenschaft, Technologische Entwicklung, Erfindung, Politik, Kultur, Sport, Gesellschaft
- Einsatz-Gesellschaftsschicht?
- Preis (schwierig vergleichbar zu sein, evtl. als Anteil von Durchschnittslohn als Vergleich)
- Zweck des Designs (Handicap, Optik (sinnlos, dekorativ, sich abheben von anderen), überwiegend von Funktion bestimmt (Platzsparen, Transport, Robustheit/Stabilität, Energieverbrauch, Tarnung, Sicherheit, günstige Herstellung, Hygiene), Umweltschutz, sozialer Gedanke,
- nachgebaut von, diente als Vorlage für ↔ wurde nachempfunden
- gezeigt auf Messe

↔	= umgekehrte Beziehung (inverse)
→	= Abhängigkeiten
{A,B.. }	= Ausprägungen/Instanzen

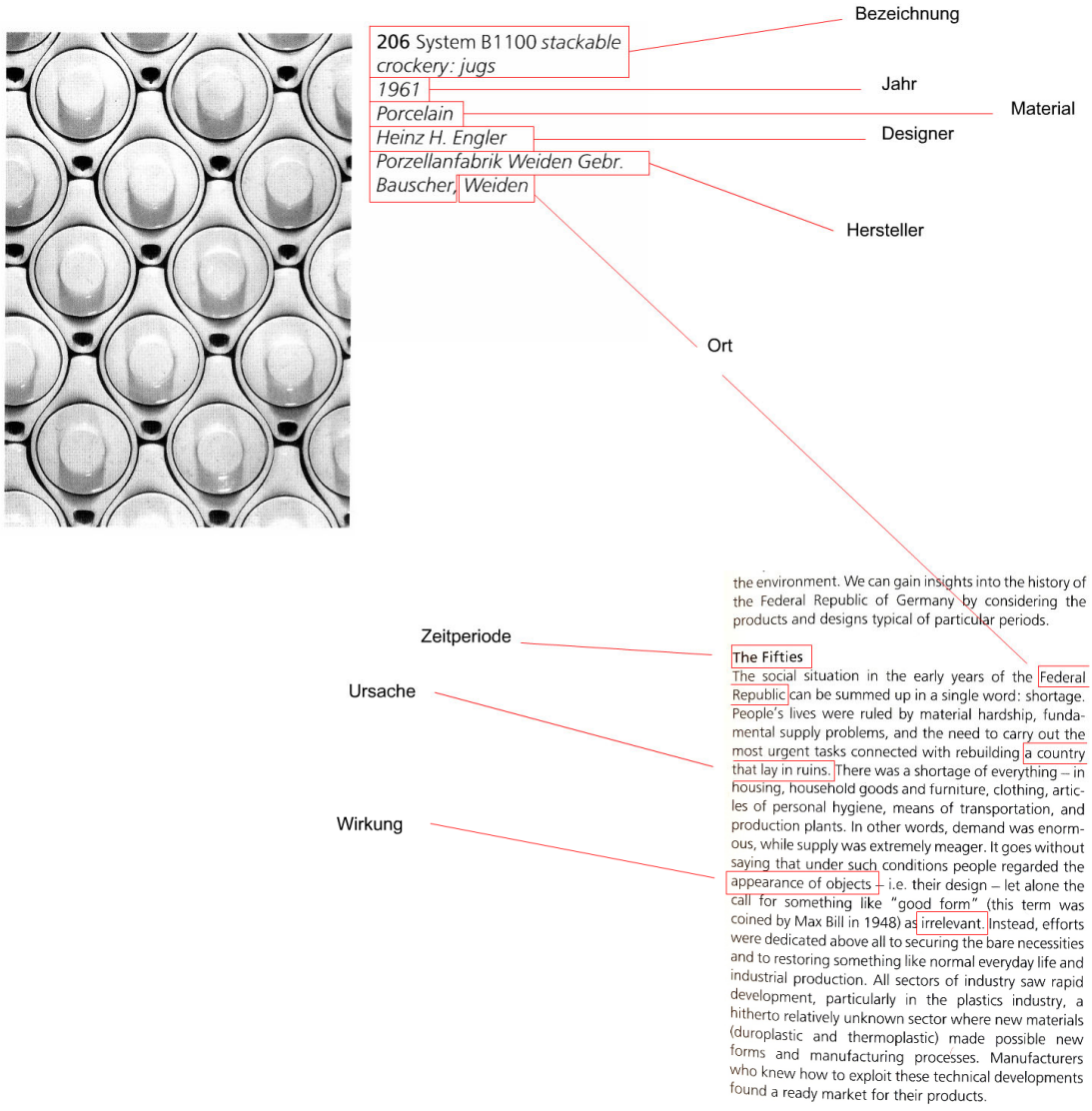


Abbildung 2 - Durchsuchen der Texte nach Begriffen, die zur Beschreibung der Domain Design wichtig sind

3.4 Erstes Ordnen

In einem zweiten Schritt wurde versucht, die Begriffe zu ordnen und herauszufinden, was sie darstellen. Unter den Begriffen fanden sich vorrangig Klassen von Dingen, aber auch Relationen. Einige Informationen ließen sich implizit aus anderen Informationen ableiten. Implizit bedeutet hier, dass die Information aus anderen Angaben geschlossen werden kann. So braucht beispielsweise nicht ausdrücklich notiert zu werden, dass ein Ereignis in den 60er Jahren statt fand, wenn ein Zeitstempel mit dem Wert 13.5.1961 12.45 Uhr gespeichert wurde.
Des Weiteren waren einige Informationen direkt primitiven Datentypen wie String, Int, Float zuordenbar.

Klassen

- Ereignis
- Zeitperiode
- Etappe / Epoche
- Gegenstand
- Stilrichtung
- Designstilrichtung
- Trend, Mode
- Gruppe
- Künstlerbund
- Künstler / Erfinder
- Hersteller (Firma)
- Herstellungsverfahren
- Ort
- Staat
- Stadt
- Ausstellung
- Museum
- Messe
- Farbe {rot, grün, blau, gelb, grau, schwarz, weiß}
- Foto
- Quellennachweis
- geographische Position
- Preis (Auszeichnung)

Rollen

- Material {Holz, Bugholz, Kunststoff, Plastik, Gummi, Metall, Papier}
- Zweck des Produkts {schneiden, backen, Sitz-Gelegenheit bieten, Schreiben}
- Zweck des Designs (Handicap, Optik (sinnlos, dekorativ, sich abheben von anderen), überwiegend von Funktion bestimmt (Platzsparen, Transport, Robustheit/Stabilität, Energieverbrauch, Tarnung, Sicherheit, günstige Herstellung, Hygiene), Umweltschutz, sozialer Gedanke
- Einsatzgebiete (Küche, Medizin, Verkehr, Wohnen, Waffen, Industrie, Kleidung)
- Einsatzgerät: {Stuhl, Tisch, Bett, Mixer, Lampe, Auto, Flugzeug, Eisenbahn, Tapete, Trinkgefäß, Maschine, Automat, Telekommunikationsgerät, Telefon, Werbung, Poster, Sportgerät, Kleidung}
- Preis
- Einsatz-Gesellschaftsschicht?
- Kontextereignisse, die die Designentwicklung beeinflussten (Fertigungsverfahren, Energie, Transport, Kommunikation, Architektur, Wissenschaft, Technologische Entwicklung, Erfindung, Politik, Kultur, Sport, Gesellschaft
- Bedeutung
  - nach eigenem Ermessen
  - nach allgemeinem Ermessen (z.B. ausgelotet durch Wikipedia)
  - nach Anzahl der Ereignisse, die davon abhängen
  - nach Anzahl der Ereignisse, die es umschließt / verallgemeinert (nur bei Zeitspannen)

Relationen

- hat Variante
- hat Vorbild in der Natur
- hat Ost-West-Zugehörigkeit
- ist public domain
- ist erstellt am
- Mensch A kennt Mensch B, B kennt A, A und B kennen sich
- hat Materialvielfalt
- hat Rundheit
- hat Beliebtheit / Aktualität heute (Zeitlosigkeit)
- hat Verbreitung / Durchsetzung
- Farbigkeit {bunt, einfarbig}
- hat Textur {gemustert}
- hat Ähnliche Produkte → Ähnlichkeit → Aussehen / Funktion
- hat Startzeit
- hat Endzeit → Dauer
- hat beteiligte Personen
- hat Ursachen (beeinflussende Dinge)
- hat Wirkungen (Dinge, die beeinflusst werden)
- Entwicklungsrichtung führt zu Ereignis
- nachgebaut von, diente als Vorlage für ↔ wurde nachempfunden
- gezeigt auf Messe

Implizite Information und bestehende Datentypen

- Jahr, Jahrzehnt, Jahrhundert
- Zeitpunkt → Datumsstempel
- Maße, Gewicht
- Bezeichnung → String
- → Dauer

↔	= umgekehrte Beziehung (inverse)
→	= Abhängigkeiten
{A,B.. }	= Ausprägungen / Instanzen

Klassen und Rollen

Viele der im letzten Abschnitt genannten Begriffe sind tatsächlich Klassen von Dingen, die im Zusammenhang mit dem Thema Design interessant sind: gestaltete Gegenstände, Firmen, Herstellungsverfahren, Ausstellungen, Preise etc. Viele Begriffe jedoch, die auf den ersten Blick als Klasse der Domain Design gelten könnten, sind nur Bezeichnungen für Rollen und somit besser als Relationen modellierbar. Erfindungsort, Hersteller und auch Material sind eher Rollen, die die Klassen „geographische Region", Firma und verschiedene Stoffe spielen. Solche Sachverhalte wurden als Relationen betrachtet. Erfindungsort wurde z.B. in der später erstellten Ontologie durch Invention takesPlaceIn GeographicalRegion umgesetzt.

Für sich stehende Objekte (*Entities*)

Unter den verschiedenen Klassen, die als Bestandteile der Domain Designgeschichte ausgemacht wurden, gibt es einige, denen das vorwiegende Interesse gilt und die in den Modellen als eigenständige Einheiten auftreten sollen. Sie werden im Folgenden als Entities bezeichnet. Für die Domain Designgeschichte sind das die Klassen

- Product bzw. Artifact
- Person
- Event

Relationen

Die meisten der als Relationen herausgefunden Begriffe konnten als ObjectProperties in OWL umgesetzt werden. Die impliziten Informationen wurden nicht aufgenommen und können von der Anwendung abgeleitet werden. Relationen zu bestehenden Datentypen wurden als DataProperties modelliert.

3.5 Verwendete Ontologie-Struktur

Alle Domain-Ontologien importieren die neu erstellte Middle-Ontologie und damit auch die modifizierte SUMO. Die Domain-Ontologien importieren sich untereinander teilweise selbst. Die folgende Graphik zeigt die owl:import-Relationen zwischen den OWL-Dateien. Grau dargestellt sind die Ontologien, die für die automatische Zuweisung von Facetten zu ihrer Entsprechung in der Visualisierung nutzbar wären[1].

[1] Auf den Mappingmechanismus wird im Abschnitt Mapping, Seite 39 eingegangen.

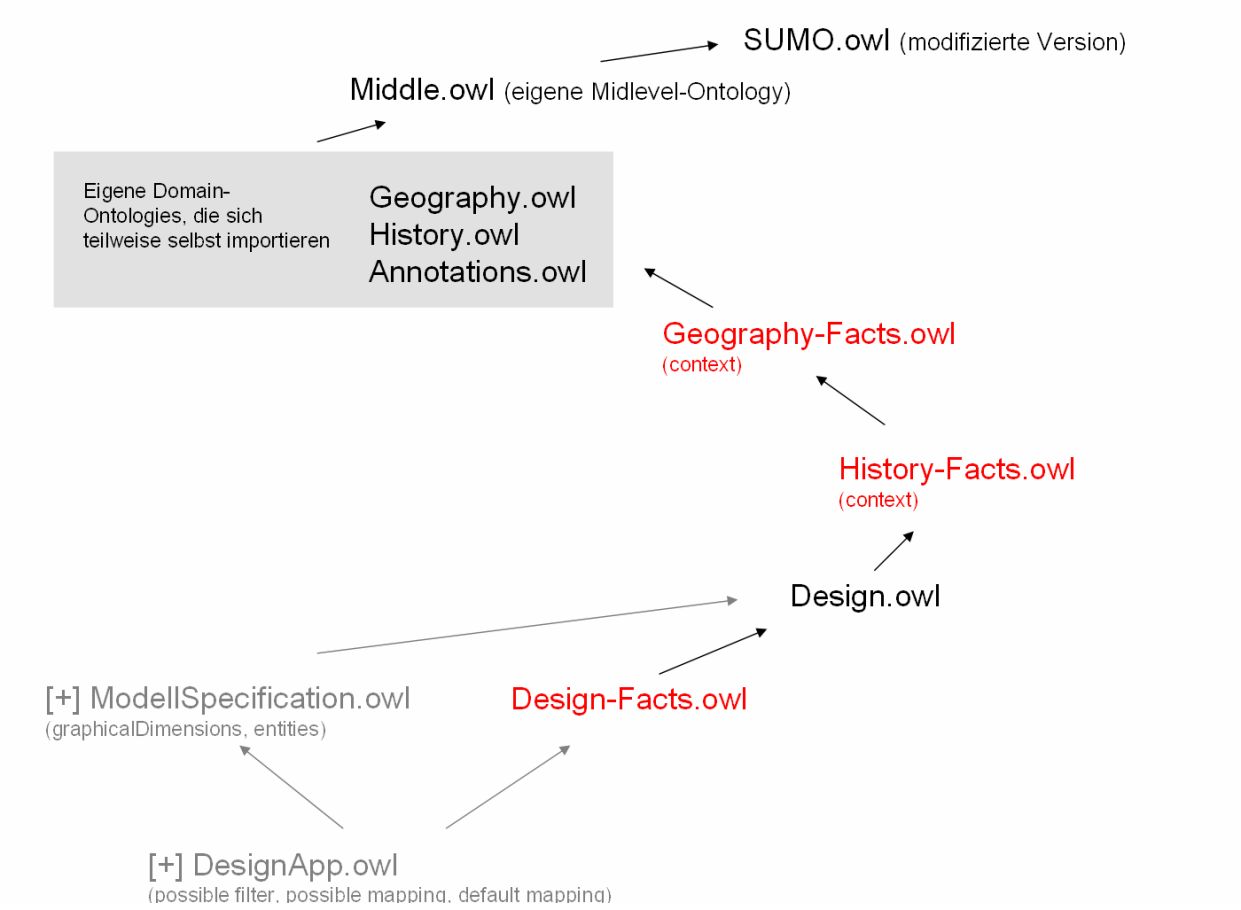


Abbildung 3 - import-Relationen der Ontologien

Übersicht

Prefix[2]	Dateiname	
sumo	Sumo.owl	reduzierte Version der Upper-Level Ontologie SUMO in OWL
mid	Middle.owl	eigene Midlevel-Ontologie mit allgemeinen Begriffen, welche die SUMO ergänzen
anno	Annotations.owl	dient der Beschreibung von Quellen (Literaturhinweise, Weblinks, Copyrightrechte …)
his	History.owl	Begriffe der Domain Geschichte (Ereignisse, Zeit)
geo	Geography.owl	Begriffe der Domain Geographie
des	Design.owl	Begriffe der Domain Design

[2] Ein Prefix wird in den OWL-Dateien verwendet, um nicht bei jeder Referenz die volle URI angeben zu müssen. sumo:Product steht für http://www.polowinski.de/ontologies/sumo.owl#Product . Die Prefixe werden in allen Ontologien und Java-Code konsistent gleich verwendet.

his-f	History-Facts.owl	Jahrzehnte, benannte Zeitspannen, geschichtliche Ereignisse…
geo-f	Geography-Facts.owl	Länder, Städte, Koordinaten von Orten …
des-f	Design-Facts.owl	Eigentliche Designgeschichte Informationen (Designereignisse, Produktbeschreibungen, Stile, Designer…)

3.6 Design-Ontologien

Ausgehend von den im vorigen Kapitel gefundenen Begriffen wurde mit Hilfe von Protégé eine Ontologie in OWL erstellt. Klassen werden dabei soweit möglich von der SUMO wiederverwertet. Ist die entsprechende Klasse nicht vorhanden, wird sie so speziell wie möglich eingeordnet. Neue sehr allgemeine Begriffe werden in der Middle.owl Ontologie notiert, so dass die Wiederverwendung möglich ist, sollte eine andere Domain visualisiert werden. Begriffe, die designspezifisch sind befinden sich in der Datei Design.owl. Die einzelnen Individuen, Instanzen von Produkten und Ereignissen beispielsweise, sind in Design-Facts notiert.
Alle Klassen, die mit dem Thema Design in Verbindung stehen, machen dies explizit, in dem sie Unterklasse von DesignRelatedSubject sind. Auf diese Weise lassen sie sich im Editor leicht auffinden, indem man in Protégé nach des:DesignRelatedSubject sucht. Für die Ontologie selbst hat diese Beziehung keine Bedeutung.

sumo:Artifact und des:DesignedArtifact

Die folgende Abbildung zeigt die Klasse sumo:Artifact, die in Unterklassenbeziehung zu sumo:CorpuscularObject steht. Durch Mehrfachvererbung wird die Erfüllung der verschiedenen Rollen (mid:Role) umgesetzt. Leider gibt es in OWL kein ausdrückliches Konzept der Rolle, weswegen dieser Weg gewählt wurde. Eleganter wäre die Verwendung einer neuen Relation erfülltRolle.

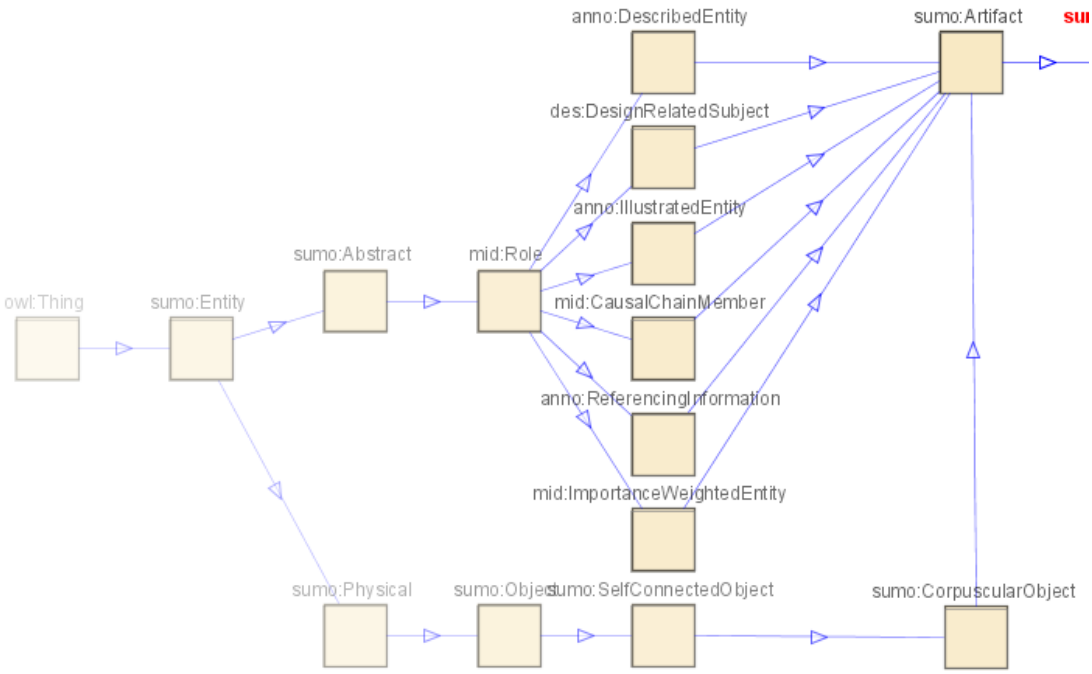


Abbildung 4 - Super Classes of Artifact

des:DesignedArtifact ist eine Unterklasse von sumo:Artifact und beschreibt Artefakte, bei denen ein bestimmtes Ziel die Gestaltung wissentlich beeinflusst hat. Alle Einträge gestalteter Dinge sind als Instanzen dieser Klasse notiert. Potenzielle Nachbarn einer Instanz der Klasse DesignedArtifact sind aus der folgenden Grafik ablesbar. Die möglichen Beziehungen einer Klasse zu einer anderen werden als die Domain-Range-Beziehungen bezeichnet.

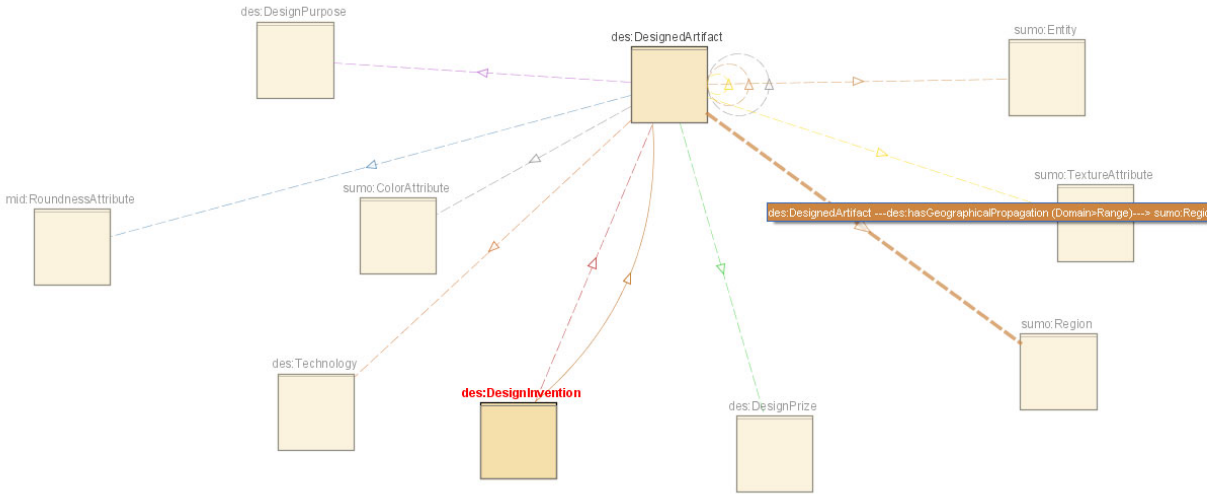


Abbildung 5 - Mögliche Nachbarschaft der Klasse DesignedArtifact

hist:Event und des:DesignEvent

Die Klasse hist:Event und des:DesignEvent sind Unterklassen von sumo:TimeInterval , also Zeitspannen. Ein Ereignis hat auf diese Weise immer eine gewisse Ausdehnung und damit eine Start- und eine Endzeit. Events sind diejenigen Entities, die vom StructureModell Helix dargestellt werden können. Weitere Beziehungen der Klasse Event sind neben hasStartTime und hasEndTime (single xsd:dateTime) [1] die Properties:

- hasInvolvedPerson (multiple sumo:GroupOfPeople)
- partyIncludedByTimeInterval (multiple sumo:TimeInterval, transitiv)
- relatedToFieldOf (mid:FieldOfSociety)
- hasAttribute (sumo:Attribute)
- hasImportanceSubjective (single: mid:RangeOfImportance)
- isEffectedBy ↔ effects (multiple mid:CausalChainMember, transitiv)
- hasImage (multiple: mid:Image)
- hasText (multiple: sumo:Text)
- takesPlaceIn (multiple sumo:Region, transitiv)

↔ = inverseProperty

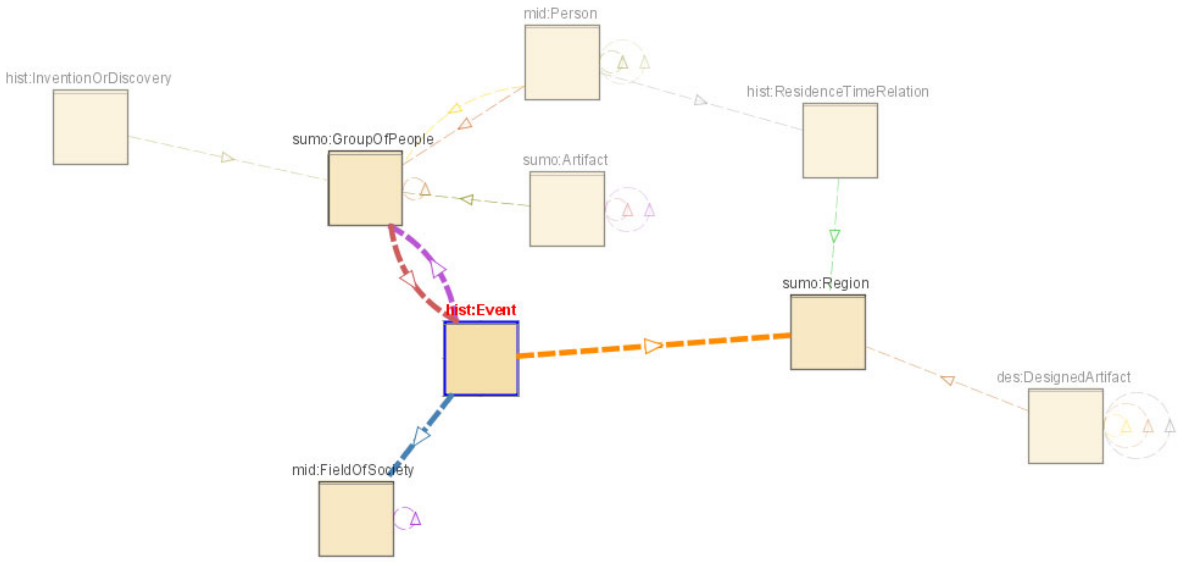


Abbildung 6 - Domain-Range der Klasse hist:Event

[1] Die Angabe in der Klammer beschreibt die Kardialität und den Range, d.h. den Zielbereich der Relation

Nachbarn der Event-Instanz BauhausFoundation

Die obere Abbildung zeigt die Instanzen, die mit der Event-Instanz BauhausFoundation in Verbindung stehen; so besitzt das Ereignis folgende Beziehungen:

- Ein Foto wurde dem Ereignis hinzugefügt: hasImage BauhausPhoto

Eine Person wurde als an der Gründung beteiligt angegeben: hasInvolvedPerson WalterGropius
Eine Stadt wurde als Ortsangabe notiert: takesPlaceIn Dessau

- Ein beschreibender Text wurde gesondert abgelegt, als Text_7, und auf diesen Bezug genommen: hasText Text_7

Außerdem ist Dessau der geopolitischen Zone Germany2006 zugeordnet. Germany2006 bezeichnet das Deutschland mit der Grenzsituation im Jahre 2006. Dies ist eine Vereinfachung – die SUMO sieht hier eine Zeit-Relation vor.
Die untere Abbildung zeigt die Beziehungen zwischen einer Erfindung, einem Event, und dem Artefakt, das erfunden wurde.

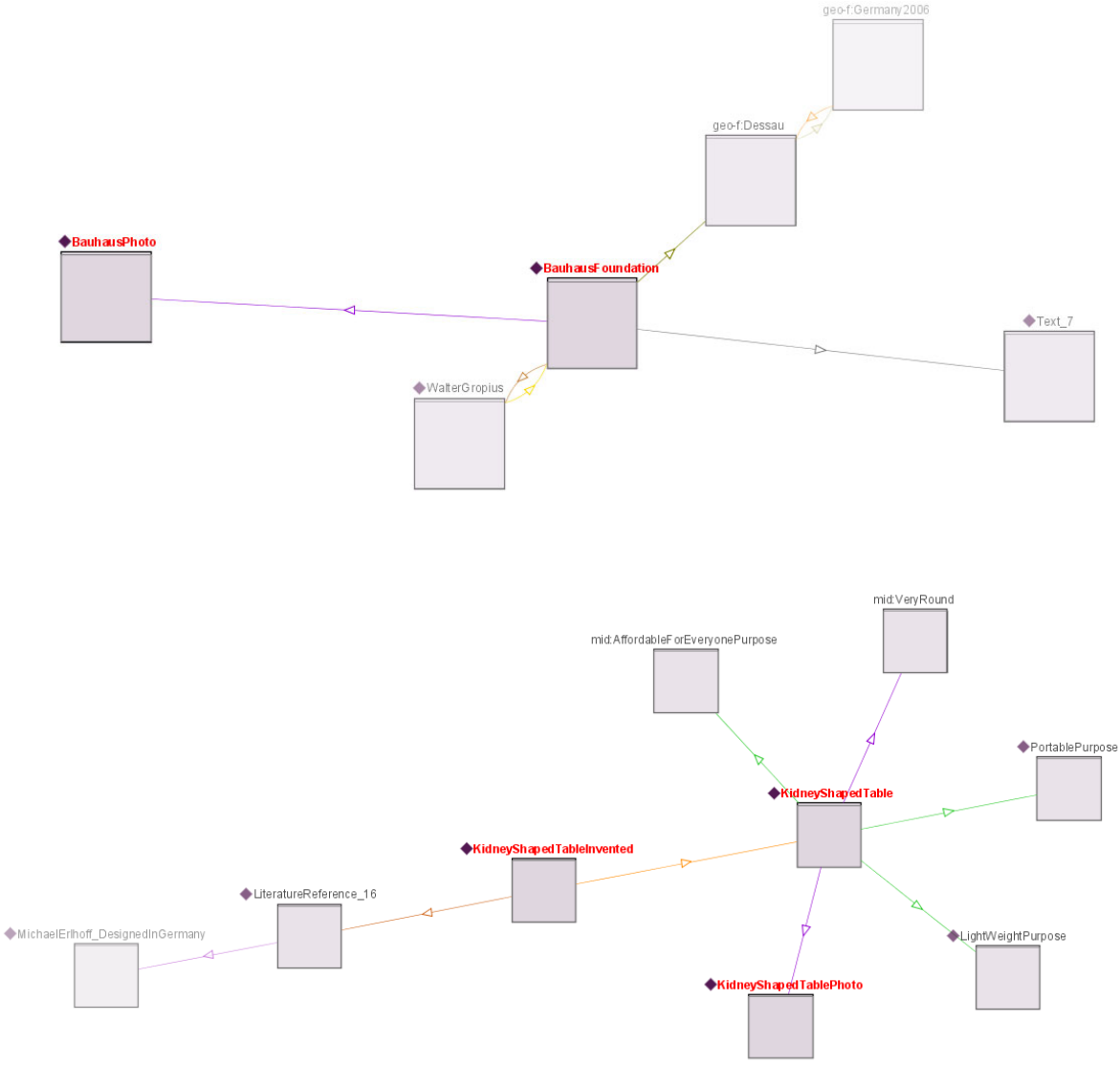


Abbildung 7 - Nachbarschaft der Instanzen BauhausFoundation und KidneyShapedTableInvented

In der zweiten Abbildung ist zu sehen, wie das Ereignis der Erfindung KidneyShapedTableInvented durch Verknüpfung mit einer anno:LiteratureReference versehen wurde, die wiederum auf ein Quellbuch Bezug nimmt.

Alle Grafiken dieses Kapitels wurden mit Jambalaya [JAMBALAYA] einem Visualisierungs-Plugin für Protégé erstellt. Für einen genaueren Einblick in die Ontologien steht eine HTML-Dokumentation zu Verfügung, die mit TopBraid™ Composer [TOP_BRAID] erstellt wurde.

3.7 Schwierigkeiten bei der Ontologiekonstruktion

Bei der Erstellung der Design-Ontologie sind einige Definitionsprobleme aufgetaucht, die oft durch Ungenauigkeiten im normalen Sprachgebrauch entstehen. Beim Ablegen des Wissens in die Wissensbasis ist man gezwungen, die Probleme explizit zu machen. Oft erscheint die Fragestellung überpenibel und haarspalterisch. Eine genaue Definition ist aber unerlässlich. Es folgen einige solcher Ontologie-Design-Entscheidungen:

- Wer sind Designer des 18. Jahrhunderts?
    - geboren und gestorben im 18. Jh.
    - bedeutende Werke im 18. Jh. entstanden?
- Was ist designed?
    - nur mit gestalterischer Intention Hergestelltes oder
    - jegliches gestaltete Ding?
- Kann ein Erfinder auch eine Gruppe sein?
- Was ist ein Museum, eine Ausstellung oder ein Gebäude?
- Gibt es punktuelle Ereignisse?
    - Fast jedes Ereignis hat eine gewisse Ausdehnung – Ausnahmen z.B.: Jahreswechsel.
    - Lösung: Ereignisse werden als Zeitspannen angesehen. Die Angabe von einer End-Zeit ist aber optional und wird in den Eingabemasken teilweise ausgeblendet.
- Wie behandelt man überlappende Zeiten und Regionen? Z.B.: Die Türkei gehört zu Asien und Europa.
    - Lösung: statt hasPart kann hier containsPartly verwendet werden. (DAG-Struktur entsteht). Es lässt sich so allerdings nicht mehr vom Land auf den Kontinent schließen.
- Können Ereignisse an mehreren bestimmten Orten (gleichzeitig) stattfinden?

Generell stellt sich bei der Ontologiekonstruktion die Frage, ob man bei der Beschreibung von neuen Klassen nicht eher mit einer Relation „hatRolle“ arbeiten sollte, als mit der Relation „ist“. Wie schon weiter oben erwähnt, ist das Konzept der Rolle nicht im OWL-Sprachumfang enthalten, so dass es auch keine Unterstützung seitens des Editors Protégé für die Verwendung von Rollen gibt.
Ein weiteres Problem ergibt sich dadurch, dass Klassen sich leider in OWL nicht als Individuen behandeln lassen. Dies gab Probleme z.B. bei Angabe von Themengebieten – hier wäre es vorteilhaft gewesen, beispielsweise die Klasse FieldOfSociety als Objekt der Relation belongsToField anzugeben, und nicht unbedingt eine Instanz davon (wie FieldOfSports). Als Work-Around wurde hier anstelle der Unterklassenbeziehung eine andere Relation (includesPartly) verwendet und alle Lebensbereiche in der gleichen Unterklassenhierarchieebene angesiedelt.
Vorteilhaft ist es auch, wenn der Editor eine Möglichkeit bietet, komplexe Formularelemente zu integrieren, um Eingaben zu vereinfachen, die sonst durch ihren formalen Aufbau sehr umständlich werden. Ein Beispiel wäre ein Eingabefeld für Bilder, bei dem die verlinkten Bilder sofort im Eingabeformular abgebildet werden und Quellenangaben notiert werden können. Der Editor Protégé lässt solche Erweiterungen beispielsweise zu und für umfangreichere Dateneingaben sollte hiervon Gebrauch gemacht werden.

3.8 Einpflegen der Daten mit Protégé

Protégé ist ein verbreiteter Editor für Ontologien, entwickelt an der Universität Stanford. Er ermöglicht die bequeme Eingabe von Instanzen über Formulare, die automatisch anhand der Ontologie generiert werden und im Layout bestimmbar sind. Weiterhin unterstützt Protégé auch bei der Verwaltung und Verknüpfung von Ontologien, unter anderem in OWL.
Die Eingabe der für die Designgeschichte-Domain wichtigen Daten, wie Startzeit des Ereignisses und Eingabe eines Titels in mehreren Sprachen sowie der Bilder, wird über passende Widgets unterstützt.
Sämtliche Fakten der Design-Ontologie wurden mit Protégé eingepflegt. Die Abbildung zeigt das Programm in der Individuals-Ansicht, in der Instanzen bearbeitet werden können.

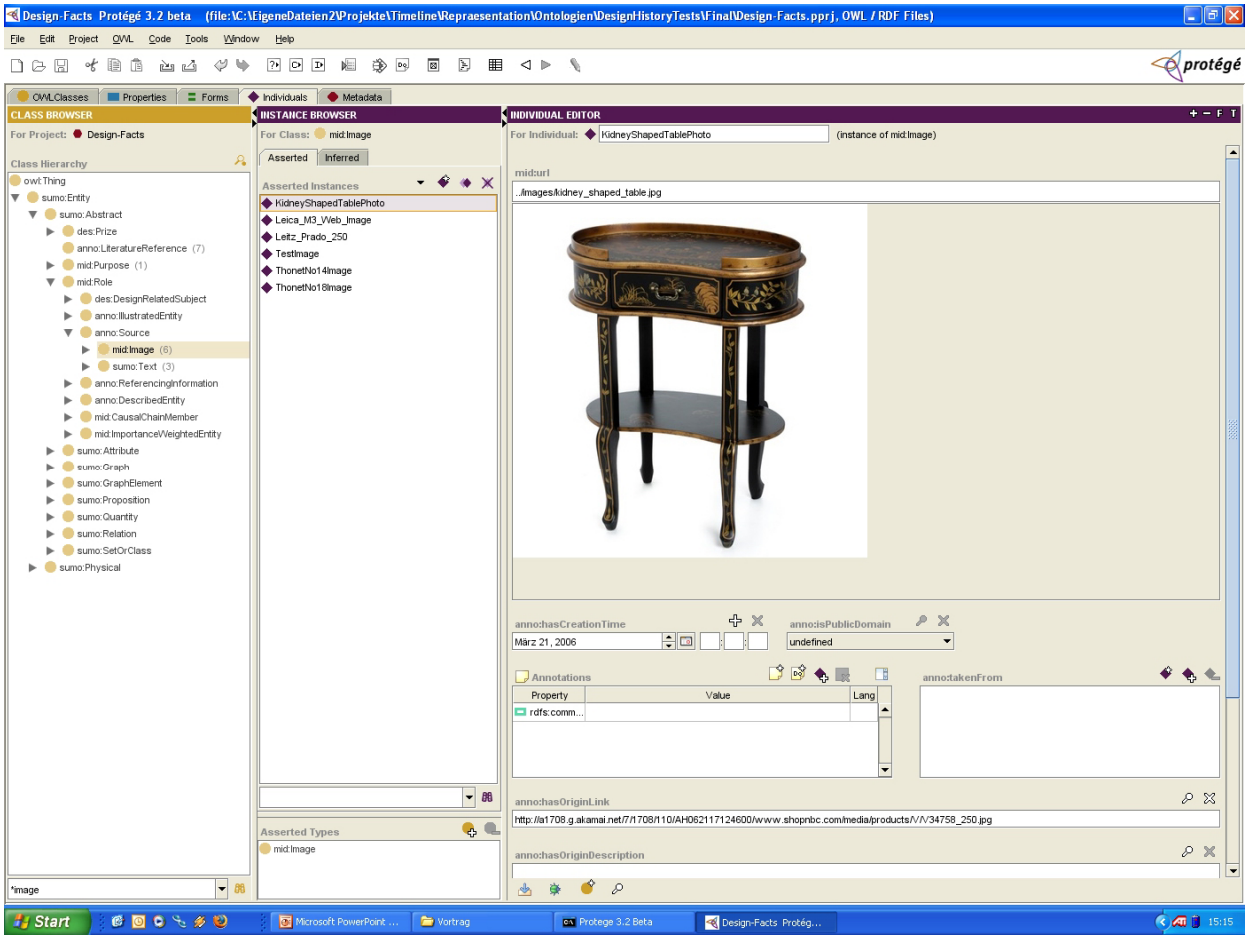


Abbildung 8 - Screenshot Protégé Instanzenansicht

3.9 Facetten

Bei der Betrachtung der Modelle aus dem Gestaltungsentwurf ist die Idee aufgekommen, das Zylindermodell für die Visualisierung von Facetten der Information zu nutzen. Im Gestaltungsentwurf ist eine beispielhafte Belegung der Seiten durch verschiedene Einsatzgebiete von Fahrzeugen vorgenommen worden. Die Überlegung ging dahin, die Seiten des Zylinder-Modells für verschiedene Facetten zu nutzen.

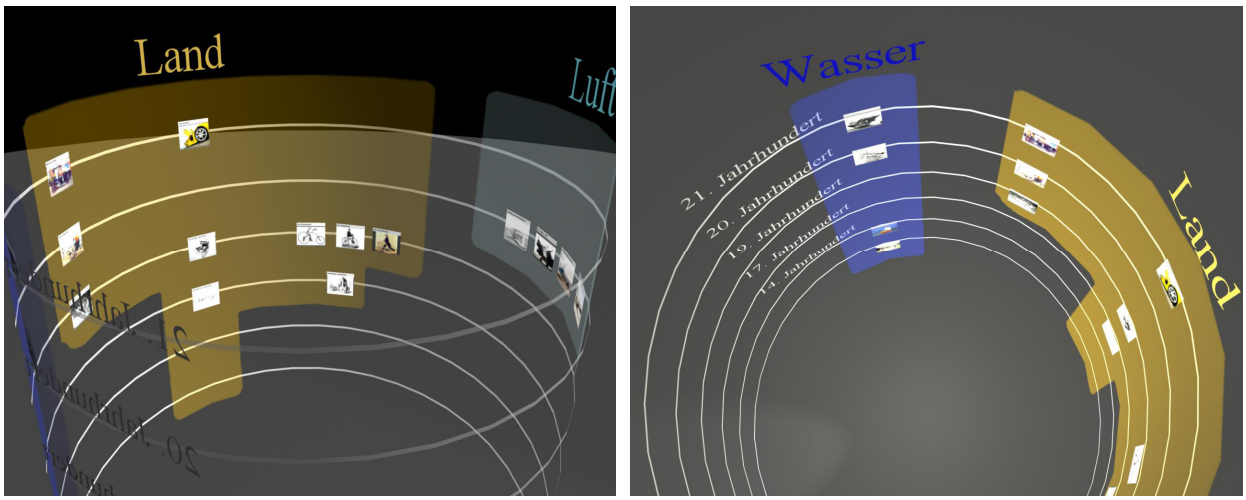


Abbildung 9 - Einteilung der Zylinderoberfläche nach Ausprägungen der Facette Einsatzort

Bei näherer Betrachtung und nach Beschäftigung mit Arbeiten zur Facettenklassifizierung [PRISS 2000, PRIETO 1991] wurde jedoch schnell deutlich, dass die Unterteilung des Zylinders nicht etwa in Facetten, sondern in Ausprägungen ein und derselben Facette bestand. Verschiedene Facetten wären mit diesem Modell alleine nicht gut darstellbar. Vielmehr sind es die Facetten, die im Gestaltungsentwurf als Dimensionen der Information bezeichnet worden waren. Ein Entity erfährt über seine verschiedenen Eigenschaften eine Einordnung in mehrere klassifizierende Systeme. Es ist also zum Einordnen eines Entities nicht der Aufbau immer komplexerer Klassifikationen nötig. Ein Beispiel:
Wünschenswert wäre die Einordnung eines Fahrzeugs in die Klasse „Grüne Landfahrzeuge des 20. Jahrhunderts." Wenn es diese Klasse nicht gibt, wohl aber die Klassen „Grüne Landfahrzeuge" und „Fahrzeuge des 19. Jahrhunderts" kann man sich entweder für eine der beiden bestehenden Klassen entscheiden, oder aber eine neue Unterklasse anlegen. Diese ist jedoch an beiden Orten gleich gut aufgehoben, da es sich bei „19. Jahrhundert" und „grün" um Ausprägungen zweier völlig voneinander unabhängiger Kriterien, zweier Facetten, handelt, nämlich der Facette Baujahr und Farbe. Wenn man die Bildung immer neuer Unterklassen verhindern möchte, kann man eine Facettenklassifikation vornehmen. Die Facettenklassifizierung wird von PRIETO folgendermaßen beschrieben:

> *„Faceted classification is synthetic. Classes are assembled by selecting predefined keywords from faceted lists. This approach provides bigger accuracy and flexibility in classification."*

Das Auffinden eines Entities kann durch Eingrenzen der Kriterien über eine Suche geschehen. Dies bietet sich besonders für virtuelle Dinge in einer elektronischen Umgebung an, bzw. war in einer physikalischen Umgebung ohne praktische Bedeutung, da es erst mit der Erfindung der elektronischen Datenverarbeitung funktioniert. Ein Buch in einer Bibliothek musste, falls es nicht mehrfach existierte und zwei neue Klassen aufgemacht werden sollten, in eine ausgewählte Kategorie einsortiert werden, in die es eher passte.
Bei der Datenvisualisierung geht es zwar nicht um die Klassifizierung von Entities, sondern um das Auslesen und Darstellen, doch bei der Visualisierung kann die Tatsache genutzt werden, dass die Facetten die Eigenschaften der Entities auftrennen und jede Facette eine bestimmte Struktur besitzt.
Die einzelnen Properties einer Klasse mit ihrem Range, der wiederum aus strukturierten Klassen bestehen kann[2], stellen Facetten der Entities dar.

[2] Eine Facette kann intern anstelle einer baumartigen auch durchaus eine DAG-artige Struktur besitzen.

Facettentypen [+]

Facetten können ihrerseits nach ihrer inneren Struktur klassifiziert werden. Abhängig von ihrem Typ können sie nur über bestimmte graphische Dimensionen abgebildet werden. Der Typ der Facette könnte genutzt werden, um eine automatische Zuordnung vorzunehmen (siehe auch Seite 39, Mapping). Diese Überlegung wird jedoch an dieser Stelle nicht weiter verfolgt.

- Kontinuierlich (z.B. die Zeit)
- Diskret
  - Ungeordnet (Menge, keine Struktur vorhanden)

    Bezeichnung
  - Geordnet (Relation zwischen den Facettenklassen, bzw. Ausprägungen einer Facette)

    Liste (z.B. hasRangeOfImportance von 1,2,3,4 oder 5)

    Baum (z.B. strenges partOf)

    DAG (z.B. Zugehörigkeit zu einem Lebensbereich, effects)

3.10 Filter

Da die Daten in einer Ontologie gespeichert sind, ist es möglich anhand aller Eigenschaften eines Entitys zu filtern. Dies ließe sich per textueller Eingabe einer SPARQL[3]-Anfrage in eine Eingabezeile erzielen, vorzugsweise sollte diese Anfrage aber graphisch über die Benutzeroberfläche zusammengestellt werden können. Beispiele für solche Anfragen wären: „Alle Produkte, die das Design eines bestimmten Gegenstandes mitgeprägt haben“ oder „Alle Erfindungen, die Umweltschutz oder Platzsparen als Zweck des Designs hatten“ (hier in natürlicher Sprache formuliert).
Die folgenden Abschnitte beschreiben, wie ein solcher graphischer Dialog aussehen könnte. Der Vorteil bei der graphischen Auswahl liegt darin, dass der Benutzer aus vorhandenen Möglichkeiten wählen kann und die Syntax der Ontologie nicht genau kennen muss. Da nun das System die Eigenschaften, die filterbar sein sollen, kennen muss, müssen diese ihm vorab per Auszeichnung mitgeteilt werden (siehe auch Abbildung Seite 40, Dynamisches Mapping- und Filtersystem):

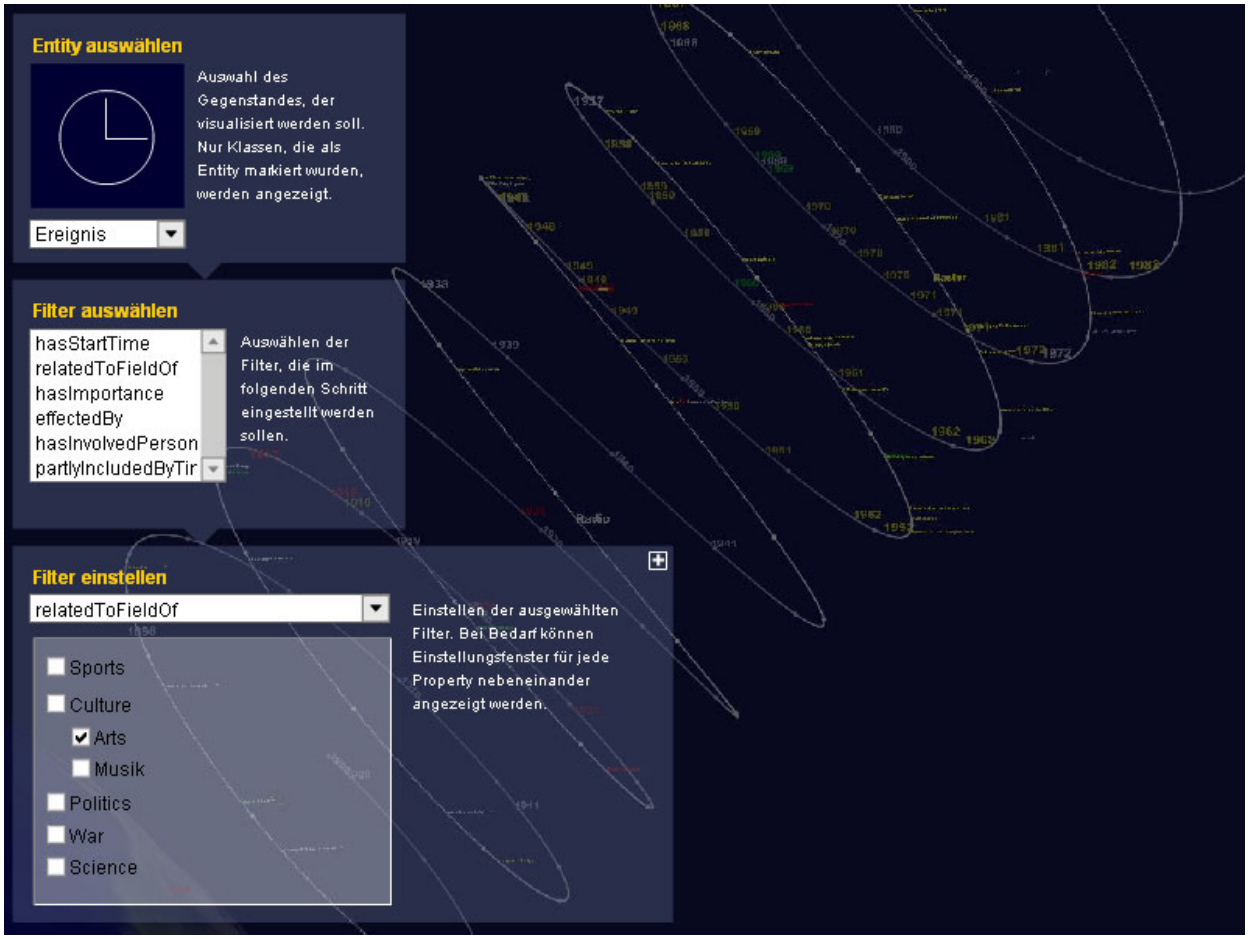


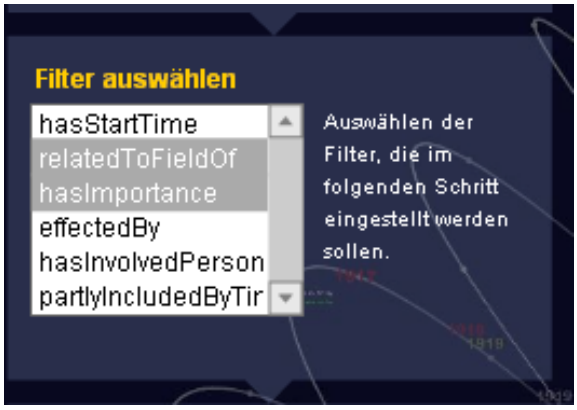
Abbildung 10 - Einstellend der Filter über das HUD

[3] SPARQL ist eine Query Language ähnlich SQL, die für die einfache Selektion der in der Ontologie gespeicherten Informationen dienen kann.

Schritt 1: Auswahl der darzustellenden Entities

Das System bietet alle als Entities gekennzeichneten[1] Klassen an. (z.B. Ereignis, Person, Produkt, Stil). Der Benutzer wählt eine der Klassen aus. Diese gilt nun als das momentan vom Modell darzustellende Entity.

Schritt 2: Auswahl der Filter



Das System zeigt alle Properties, welche das ausgewählte Entity besitzt, und die als FilterableProperty ausgezeichnet sind.
Der Benutzer markiert alle Properties, deren Werte bei der Filterung der Entity-Instanzen berücksichtig werden sollen. Bei allen anderen, den nicht markierten Properties, kann jeder Wert für die Property vorliegen.

Schritt 3: Einstellen der Filter

Das System zeigt für jede markierte Property eine Auswahlmöglichkeit, um den Range der Property weiter einzuschränken. Die graphische Repräsentation dieser Auswahlmöglichkeit ist abhängig vom Typ der Property. Für ObjectProperties können Bäume verwendet werden – für DataProperties Schieberegler (Int, Float), Checkboxen (boolean) etc., Eingabefelder für reguläre Ausdrücke und Strings. Bei der Auswahl sollte möglichst auch eine AND-Verknüfung der einzelnen Klassen und Ausprägungen einer Facette möglich sein.

[1] Die Kennzeichnung von Entities und FilterableProperties kann über eine weitere Ontologie erfolgen.

4 Datenvisualisierung

Die Datenvisualisierung bezeichnet jede graphische Repräsentation von Daten und hat somit zwar seit der Erfindung des Computers mehr Bedeutung erlangt, aber auch vorher schon seit langem in Form von Karten, Graphen und Diagrammen existiert.
Die komplexe Aufbereitung rein abstrakter Zusammenhänge aus dem Bereich der Wirtschaft, z.B. die Aufbereitung von Unternehmensdaten, sind Thema komplexer Lösungen, wie etwa VisualMining [VISUAL_MINING] oder Eye-Sys™ [EYE_SYS], die auch aufwändige 3D-Grafik einsetzen.
Auch in der Softwareindustrie verlangt die Entwicklung immer komplexere Programme, intuitiv begreifbare Übersichtsdarstellungen, um, ohne den Quellcode lesen zu müssen, einen Überblick über Zusammenhänge und Struktur der Software zu bekommen. Dreidimensionale graphische Darstellungen sind hier bisher selten. Im Bereich der zweidimensionalen Darstellung durch UML Diagramme oder „Outliner“ [1] der Entwicklungsumgebungen zeigt sich jedoch, dass alternative Ansichten von Software wesentlich zur Verbesserung der Arbeit beitragen können. Eine gute Übersicht zur Softwarevisualisierung findet sich in [ZEHLER 2004].

4.1 Visualisierung zeitlicher Daten

Die Visualisierung historischer Daten ist bislang oftmals auf Zeitstrahlen begrenzt und kein Gegenstand von 3D-Visualisierungen. Als Möglichkeiten der Visualisierung solcher Daten gibt es als traditionelle, interaktive Medien den Kalender, den Zeitstrahl und als nicht interaktives Medium den Film.
Eine verbreitete Darstellungsform für chronologische Daten ist der Zeitstrahl. Bei der Papierform ist es jedoch notwendig, sich für ein bestimmtes Detail-Level zu entscheiden. Wird dies ausreichend groß gewählt, so dass Details erkennbar sind, so passt der ganze Zeitstrahl häufig nicht mehr auf eine Seite und muss über mehrere Blätter verteilt werden. Der Überblick geht somit verloren.
Bei der digitalen Darstellungsform hat man die Möglichkeit des unendlichen Zooms und damit einen Sachverhalt, der über die Nachbildung der Realität hinausgeht. Der Computer ahmt hierbei nicht ein schon bestehendes Medium wie das Buch nach, sondern nutzt seine zusätzlichen Fähigkeiten.

4.2 Hyperhistory

Die Website Hyperhistory (Siehe Abbildung 11) ist eine umfangreiche Chronologie, die statische Zeitstrahlen einsetzt, um Geschehnisse dazustellen [HYPERHISTORY]. Über Schaltflächen lässt sich zwischen verschiedenen Ansichten, wie punktuellen Ereignissen, längeren Zeitspannen in der Geschichte, z.B. dem Bestehen von Weltreichen, und Lebenszeiten von einflussreichen Personen blättern. Außerdem gibt es eine Sammlung von geschichtlichen Karten. Für die Ansicht Personen und Ereignisse wird eine Farbkodierung verwendet, die zur Unterscheidung der Zugehörigkeit zu verschiedenen Lebensbereichen, wie Wissenschaften, Kultur, Religion und Politik dient. In den anderen Ansichten dient die Farbe lediglich der Unterscheidung und eine einzelne Farbe bekommt keine Bedeutung zugewiesen. Die Probleme der statischen Darstellung werden über Buttons zum Aufklappen verfeinerter Darstellungen und über sehr kleine und klare Schrift zu lösen versucht. In der Personenansicht wird eine Aufteilung des Lebensbereichs Kultur in Kunst, Musik und Literatur vorgenommen. Die X-Achse wird bei allen Diagrammen für den Zeitverlauf genutzt. Auf der Y-Achse wird bei der Zeitspannendarstellung eine Einteilung in die Kontinente vorgenommen. Bei den anderen Darstellungen findet in Y-Richtung zusätzlich zur Farbko-

[1] Ein Outliner stellt eine gröbere Beschreibung einer komplexen Struktur dar. In der Entwicklungsumgebung Eclipse repräsentiert der Outliner über eine symbolische Darstellung Attribute und Methoden einer Klasse, ihre wesentlichen Merkmale und die Reihenfolge im Quellcode.

dierung eine Sortierung nach Lebensbereichen statt. Verschiedene Darstellungen lassen sich bei Hyperhistory nicht überlagern. Ein Vergleich kann durch Blättern zwischen den Ansichten erreicht werden.

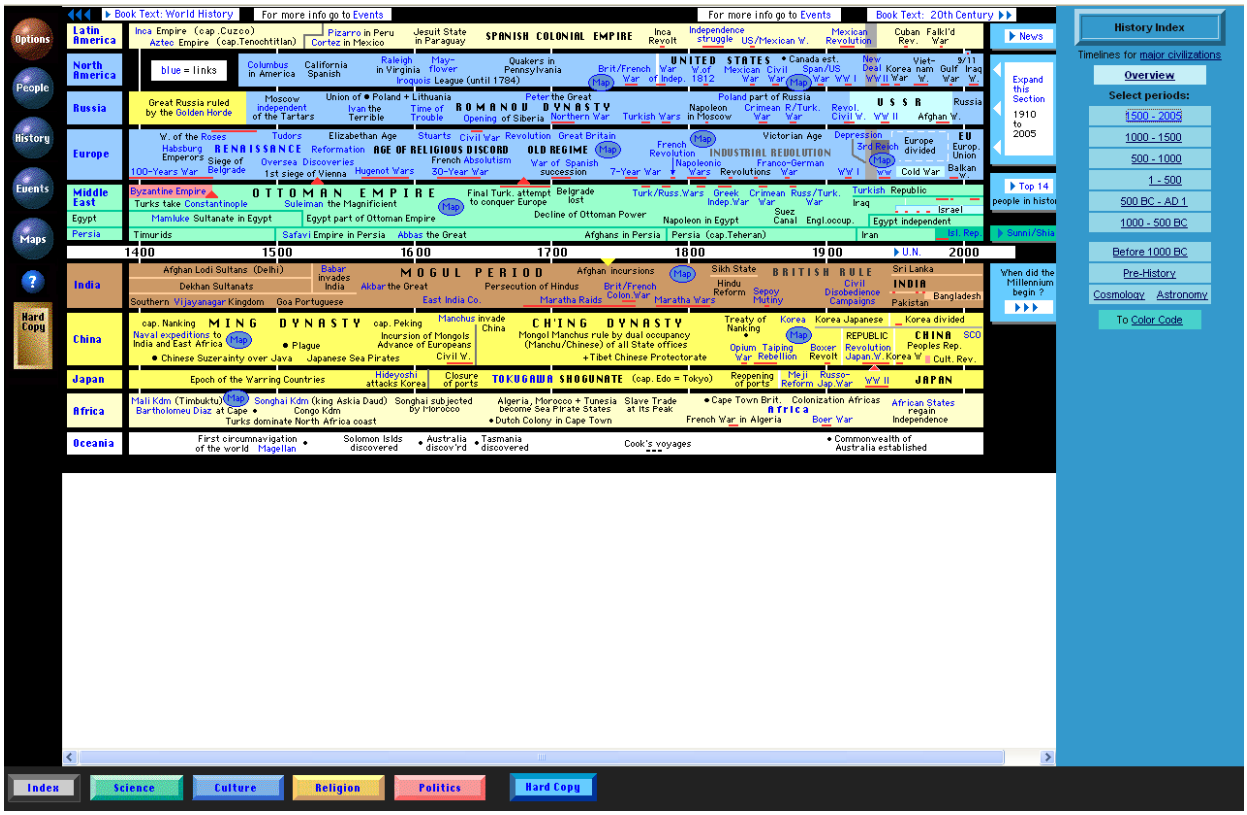


Abbildung 11 - Hyperhistory

Dynamic Timelines

Eine umfangreiche Arbeit zur dynamischen Darstellung von Zeitstrahlen ist das Projekt Dynamic Timelines am MIT Media Laboratory [KULLBERG 1996]. Es werden hier anhand der Domain Photographie verschiedene Techniken erprobt, um den Gegensatz zwischen Detail und Übersicht zu lösen, sowie verschiedene Belegungen der räumlichen Dimensionen mit Bedeutung ausgetestet. Ebenfalls werden Mechanismen zur Darstellung von hoher Ereignisdichte und zur Stabilisierung des Raumes getestet. An gegebener Stelle wird ein Vergleich zu den Ergebnissen von KULLBERG gezogen.

Die folgende Abbildung ist ein Screenshot der Visualisierung von Lebensspannen wichtiger Photographen. In X-Richtung verläuft die Zeit, in Z-Richtung wurde eine geographische Auftei-lung in Staaten vorgenommen. Nähert man sich einem Balken, der eine Zeitspanne repräsentiert, so werden auf ihm Details eingeblendet.

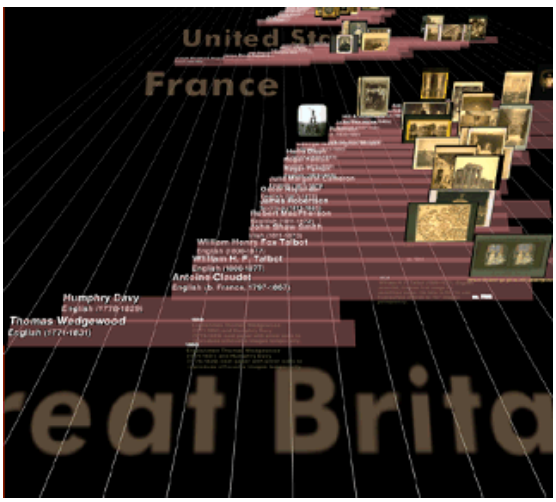


Abbildung 12 - Dynamic Timelines

4.3 Graphisches Vokabular - graphische Dimensionen

Der Begriff graphisches Vokabular geht zurück auf Thomas Zehler [ZEHLER 2004], der diesen Begriff in einem Report zur Software-Visualisierung verwendet. Er steht für Eigenschaften der graphischen Darstellung, die sich als Vokabular verwenden und – teilweise – mit Bedeutung belegen lassen.
Im Framework verwende ich stattdessen den Begriff graphische Dimensionen, der dasselbe meint, aber in der Einzahl leichter zu verwenden ist. Eine graphische Eigenschaft ist eine graphische Dimension.

ZEHLER nennt für 2D-Darstellungen folgende visuelle Variablen des graphischen Vokabulars:

- Position auf der Ebene
- Größe (Länge, Fläche, Volumen)
- Helligkeitswert
- Musterung, Textur
- Farbe
- Richtung oder Orientierung
- Form des Elementes

Für die Darstellung im dreidimensionalen Raum und durch die Verwendung von Strukturen kommen zusätzliche Dimensionen hinzu, die in Abbildung 13 aufgelistet sind.

Einteilung der graphischen Dimensionen zur Visualisierung von Software

Sucht man Kriterien zur Abbildung von Softwareelementen und Situationen bei der Visualisierung von Software auf graphische Dimensionen, so lässt sich nach ZEHLER folgende Einteilung verwenden:

Nominal (Benennung und Unterscheidung von Dingen)

- Farbe, Fläche, Textur, Form, Transparenz, Sichtbarkeit, Vordergrund, Hintergrund und Tiefe

Ordinal (Darstellung der Reihenfolge von Dingen)

- Helligkeit, Sättigung, Grauton, Länge, Größe, Bewegung und Kontrast

Quantitativ (Angabe von Größenverhältnissen)

- Position, Orientierung, Winkel, Länge und Distanz.

Diese Einteilung ähnelt der Bildung von Facettentypen (siehe Seite 29ff.). Beide Einteilungen bieten eine Grundlage für die Einschränkung der freien Belegung der graphischen Dimensionen mit Bedeutung.

Einteilung der graphischen Dimensionen anhand ihrer Unabhängigkeit

Abbildung 13, graphische Dimensionen, veranschaulicht die wichtigsten graphischen Eigenschaften am Beispiel des Helix-Modells. Die grau hinterlegten Eigenschaften bilden eine Untermenge von untereinander nicht orthogonalen, d.h. nicht vollständig unabhängigen Eigenschaften. Die Gruppen von Eigen-

schaften hingegen sind vollständig orthogonal zueinander – die Veränderung einer Eigenschaft aus einer Untermenge lässt die Eigenschaften einer anderen Untermenge vollständig unbeeinflusst.
Ein Beispiel hierzu: Größe und Farbe lassen sich vollständig unabhängig voneinander verändern. Form und Orientierung hingegen beeinflussen sich gegenseitig. Eine Änderung der Form kann die Orientierung, da sie sich dem Betrachter über die Form erschließt, leicht verändern. Ebenso kann dies umgekehrt geschehen. Auch Transparenz, Helligkeit und Sättigung stehen über die Farbe miteinander in Verbindung, denn nur die Farbe wird letztendlich vom Bildschirm dargestellt. Beim Fehlen von durchscheinendem Hintergrund als Referenz können transparente Flächen wie helle opake Flächen erscheinen.

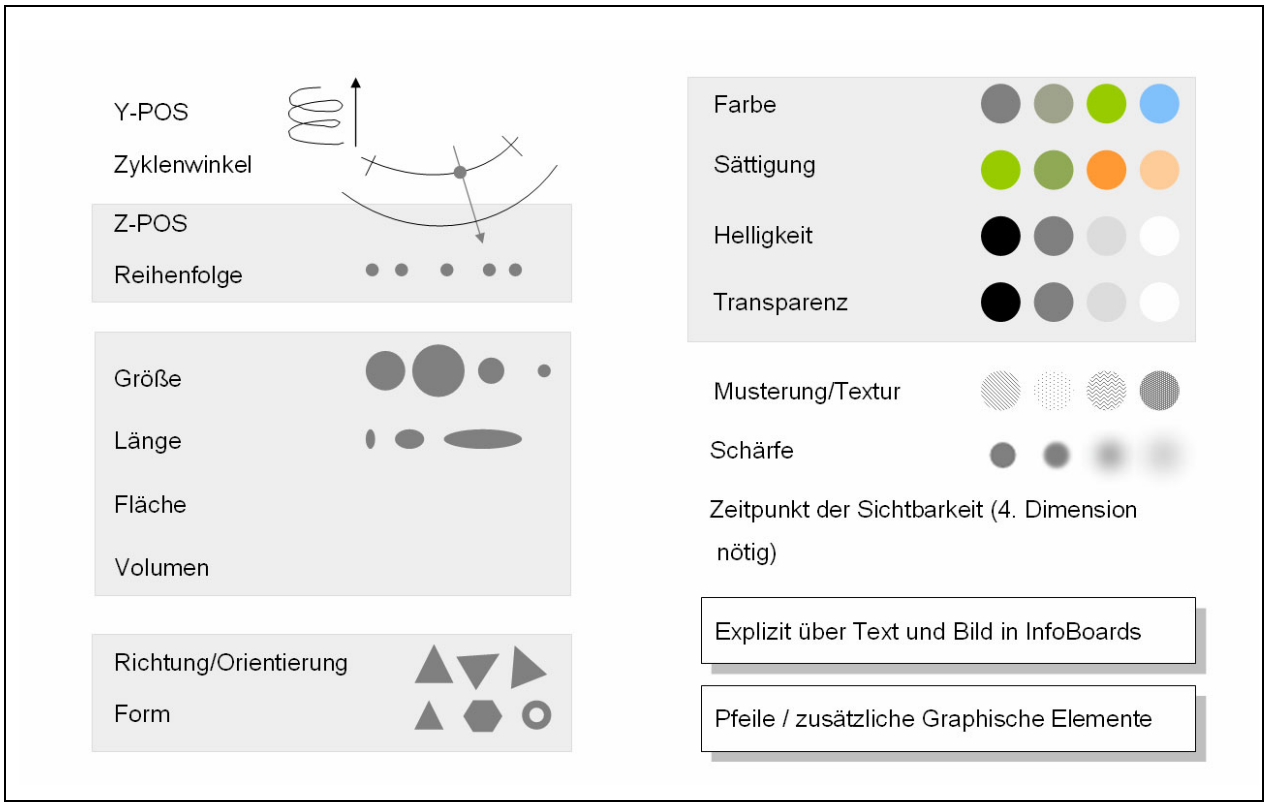


Abbildung 13 – Graphische Dimensionen des Helix-Modells

Implizite vs. Explizite Darstellung

Neben den graphischen Dimensionen, die sich durch die Eigenschaften der darzustellenden Objekte ergeben, kann Information natürlich zusätzlich explizit über Text und Bild dargestellt werden. Die explizite Darstellung über Text ist zwingend für Bezeichnungen wie Namen, kann aber auch zusätzlich zur impliziten Information angeboten werden. Beispielsweise ist es sinnvoll, ein Datum notfalls auch bequem ablesen zu können, anstatt anhand der Position das Datum schätzen zu müssen. Implizite Informationen sind schneller und intuitiver erfassbar, liefern großen Mehrwert durch den Kontext, sind aber oft unscharf. Durch die geschickte Kombination beider Darstellungsweisen lässt sich die Forderung nach schnellem Überblick und gleichzeitiger Möglichkeit, ins Detail zu gehen, unterstützen. Mehrere Textinformationen können vom Strukturelement InfoBoard[1] aufgenommen werden. Zusätzliche Elemente im Bildraum können genutzt werden, um z.B. Relationen darzustellen. Relationen sind Beziehungen „zwi-

[1] InfoBoards bezeichnen komplexe, aus Text, Bild und graphischen Primitiven zusammengesetzte Objekte (siehe auch Seite 64).

schen“ zwei Dingen. Eine Darstellung über ein zusätzliches graphisches Element ist also nahe liegend (siehe auch Seite 69, Relationen).

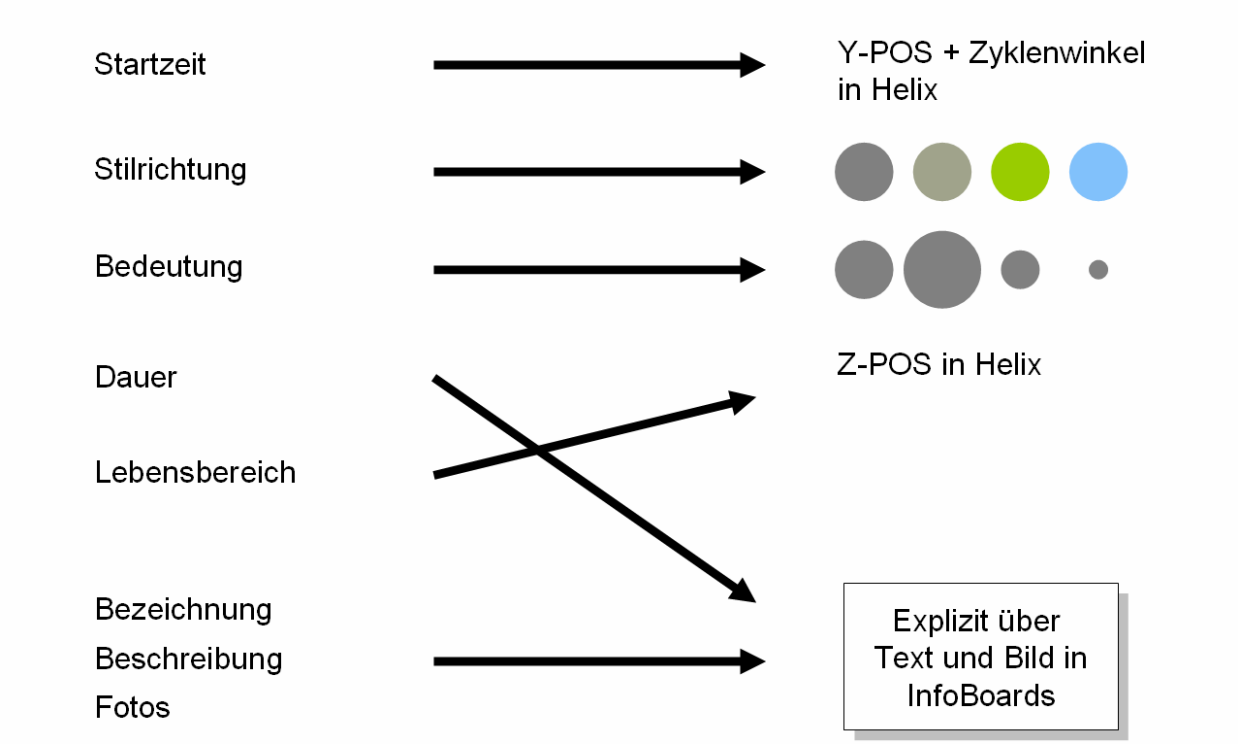


Abbildung 14 – Mögliche Abbildung der Facetten auf graphische Dimensionen am Beispiel des Helix-Modells

Abbildung 14 – Mögliche Abbildung der Facetten auf graphische Dimensionen zeigt eine mögliche Zuordnung der Facetten auf die graphischen Dimensionen des Helix-Modells. In der Beispielimplementierung wurde jedoch eine wesentlich simplere Abbildung gewählt, Bedeutung, Stilrichtung und Dauer wurden bisher nicht abgebildet:

Startzeit → Y-POS + Zyklenwinkel
Lebensbereich → Farbe
Bezeichnung, Beschreibung, Ort, Bild → Text im InfoBoard

4.4 Mapping

Das Mapping betrifft zum einen die Auswahl eines geeigneten Modells für das gewählte Entity, zum anderen die Abbildung von Facetten dieses Entitys auf das graphische Vokabular des Modells. In diesem Abschnitt wird ein flexibler Mapping-Mechanismus entworfen. In der Beispielimplementierung findet sich stattdessen bislang eine statische Zuordnung über den Programmcode (siehe Seite 53, Mapping im Framework).

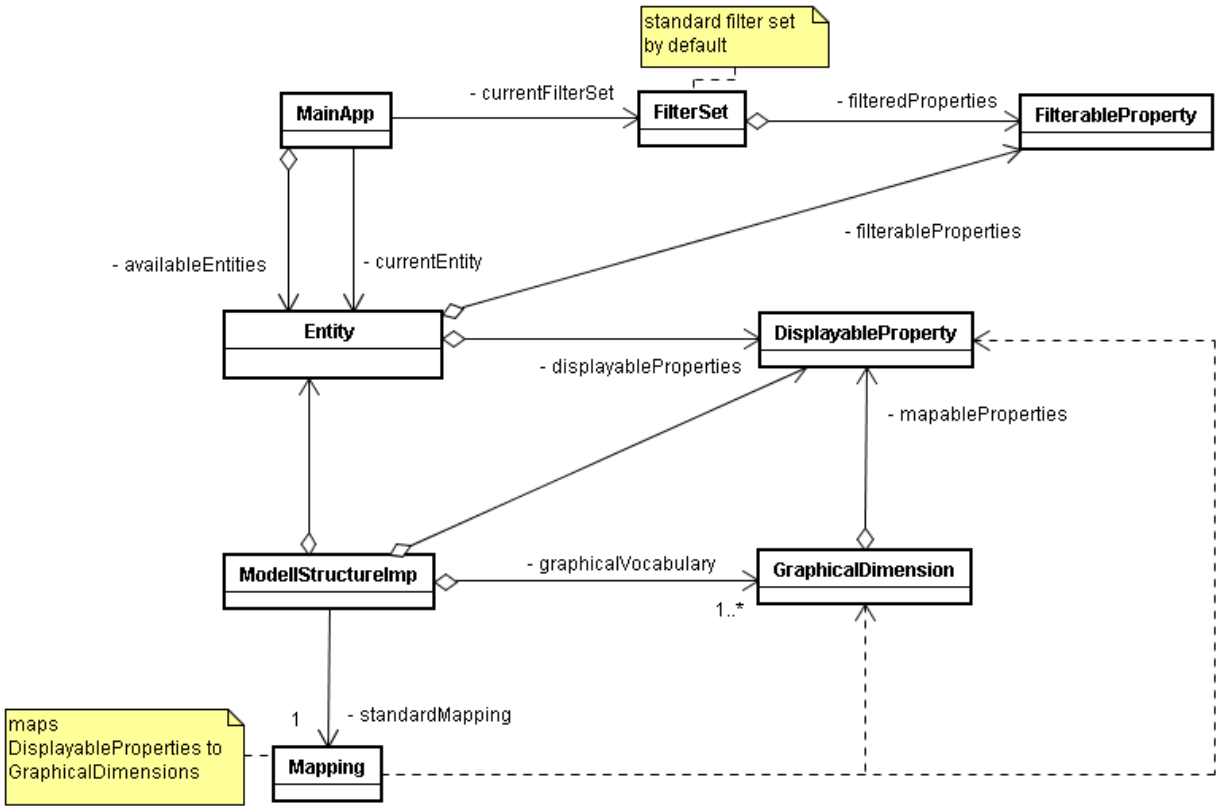


Abbildung 15 - Dynamisches Mapping- und Filtersystem

Auswahl eines geeigneten Modells [+]

Nicht jedes Modell kann für jedes Entity verwendet werden. Vielmehr gehört zu der Entwicklung eines neuen Modells eine Spezifikation, z.B. in Form einer weiteren Ontologie, die angibt, welche Entities das Modell darstellen kann und welche Properties gesetzt sein *müssen*. Die Zeit-Helix würde z.B. beschreiben, dass sie Events darstellen kann und hasStartTime nötig ist, denn sie benötigt das Vorhandensein von hasStartTime, um ihre Struktur zu generieren.
Es sollte bei der Spezifikation möglichst auf Begriffe aus Upper-Level-Ontologien Bezug genommen werden.
Es ist jedoch möglich, dass mehrere Modelle für die Darstellung eines Entitys nutzbar sind. In dem Fall bietet das System eine Auswahlliste der möglichen Modelle an, aus denen der Nutzer wählen kann.

Mapping von Facetten auf graphische Eigenschaften des Modells

Spezifikation

Jedes Modell besitzt ein anderes graphisches Vokabular und somit eventuell auch eine andere Mächtigkeit, mit der verschiedene Facetten des Entities wiedergegeben werden können. Das graphische Vokabular wird über eine Liste von GraphicalDimensions bestimmt, die das Modell per Spezifikation zugewiesen bekommt.
Jede der GraphicalDimensions besitzt eine Liste von Properties, die auf sie abgebildet werden können. Für jede GraphicalDimension eines Modells muss angegeben werden, welche Properties in Frage kommen. Zusätzlich wird für jedes Modell ein Standard-GraphicalDimension-Property-Mapping spezifiziert.

Zuordnung vom Nutzer

Dieses Standard-Mapping kann vom Nutzer, soweit es die GraphicalDimensions erlauben, verändert werden. Alle Properties der Entity, die einer GraphicalDimension des Modells zugeordnet sind, werden vom System aufgelistet. Der Nutzer kann nun im Rahmen der oben spezifizierten Möglichkeiten auswählen, welche graphische Dimension des Modells der jeweiligen Property zugeordnet werden soll.
Das Mappingsystem kann ähnlich dem Filtersystem gestaltet werden und sollte zur Laufzeit veränderbar sein.

Abbildung 16 - Einstellung des Mappings über das HUD

5 Framework und Gestaltung des Mediums

Dieses Kapitel dokumentiert den konkreten softwaretechnischen Entwurf der Architektur und wesentlicher Aspekte des Frameworks. Zugleich werden parallel mediengestalterische Überlegungen beschrieben. Es wird pro Abschnitt jeweils auf ein Gestaltungsproblem eingegangen und die entsprechende Frameworkklasse erläutert, welche die Problematik löst, soweit die bisherige Implementation dies leistet. Über die Implementation hinausführende Abschnitte sind durch das Symbol [+] gekennzeichnet.

5.1 Technologien und Werkzeuge

Jena

Jena bietet eine API, um mit Java auf Ontologien zuzugreifen und wird vielfach verwendet, unter anderem auch im Ontologie-Editor Protégé. Da die OWL-Dateien mit Protégé geschrieben sind, lassen sie sich problemlos über die Jena-API wieder auslesen. Momentan wird die Datei-Variante verwendet, d.h. die Ontologie wird aus Textdateien in XML-Notation ausgelesen. Jena unterstützt jedoch auch die Möglichkeit auf OWL per Datenbank zuzugreifen, was bei wachsendem Umfang der Datenmenge genutzt werden sollte.

Protégé

Als Editor wurde der frei verfügbare Editor Protégé eingesetzt, der vor allem bei der Eingabe der Instanzen sehr hilfreich ist (siehe auch Seite 29, Einpflegen der Daten mit Protégé).

Java3D

Warum Java3D

Eine 3D-API in Java ließ sich leicht mit der Jena-API kombinieren, die für die Abfrage der Ontologie verwendet werden sollte und ermöglicht es, die auf das Framework aufbauenden Anwendungen auch über das WWW verfügbar zu machen.
Eine Recherche verschiedener 3D-APIs ergab, dass Java3D über alle benötigten Fähigkeiten verfügte und als API für wissenschaftliche Visualisierungen empfohlen wird. Der Vorteil, der sich durch den hohen Abstraktionsgrad ergibt, wiegt die Performanceeinbußen gegenüber direkter Implementierung in bspw. OpenGL [OPENGL] auf, zumal komplexe Geometrie nicht zu erwarten war, denn die Grundformen der Visualisierung sind Rechtecke und Texte.

Szenengraph

Ein weiterer Grund für die Verwendung von Java3D war die Bedingung, eine Scenegraph-API zu erhalten. Diese liefert eine baumartige Struktur der Szenenelemente. Das Bestehen eines Szenengraphen ist zum Beispiel beim Picking, der Berechnung der Elemente, auf die ein Maus-Klick ausgeführt wurde, hilfreich.
Von der MainApp Anwendung wird folgender grober Szenengraph aufgebaut:

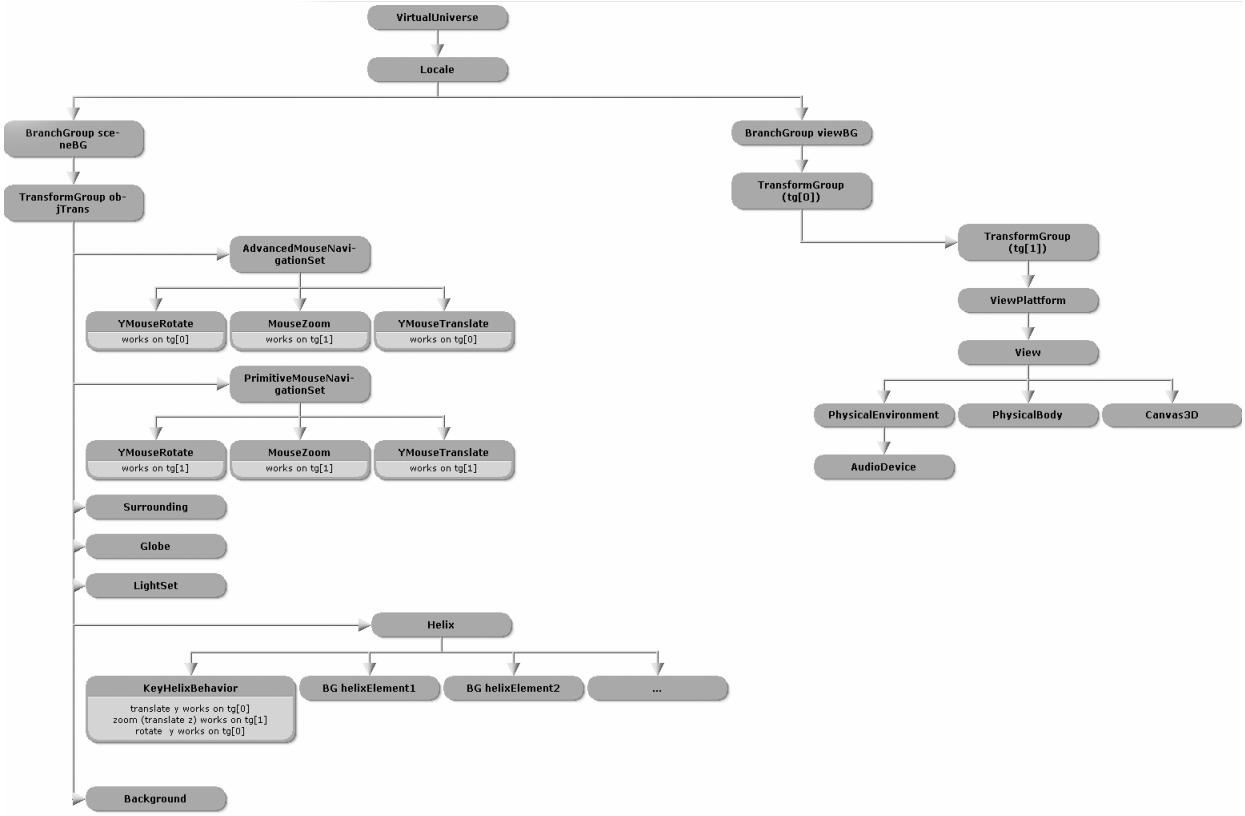
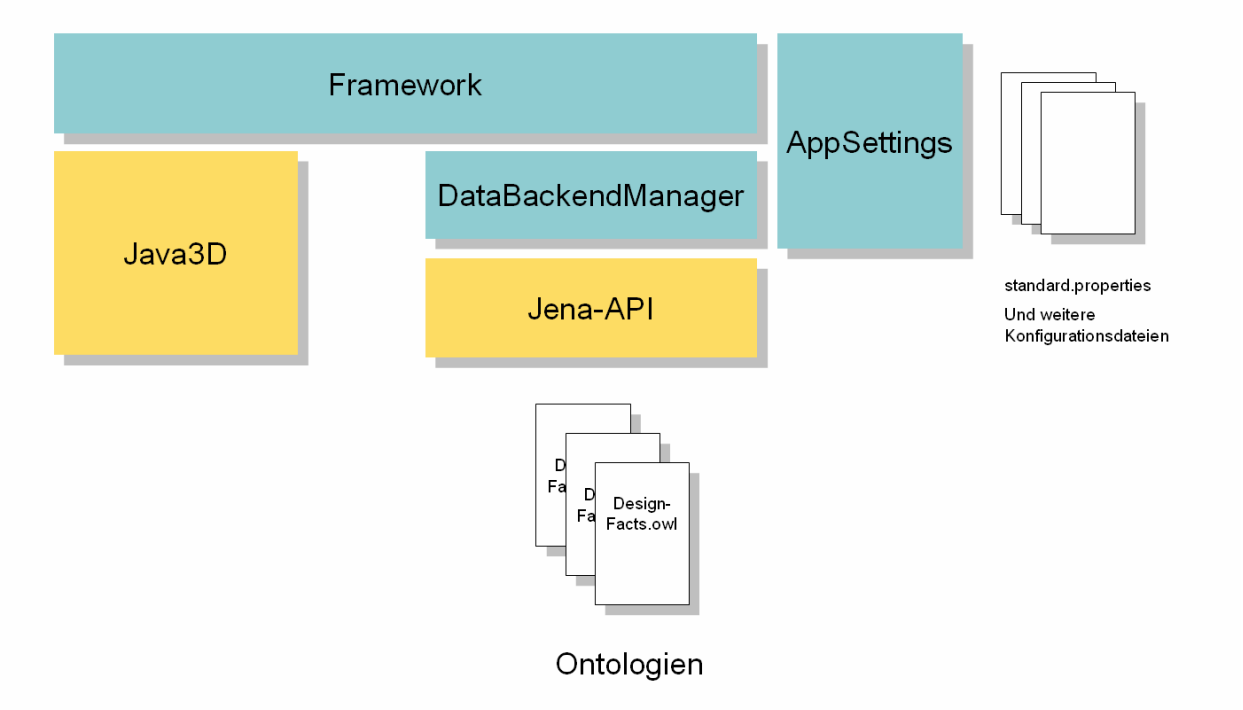


Abbildung 17 - Szenengraph MainApp

5.2 Architektur

Dieser Abschnitt soll einen Überblick über den Aufbau des Frameworks und die Hauptklassen geben. Für eine genaue Beschreibung der Klassen und der wichtigen Methoden wurde die Javadoc ausführlich eingesetzt.



Das Framework greift über den DataBackendManager auf die Jena-API zu, welche die Ontologien anspricht (Facade Pattern). Zum Erzeugen der 3D-Grafik bedient sich das Framework Java3D. Momentan werden direkt die Java3D Klassen erweitert. Neue Klassen können so leicht in den Szenengraphen eingebunden werden. Wünschenswert wäre hier die Verwendung einer Factory um das Framework unabhängig von Java3D zu machen. Über das Singleton AppSettings wird die aktuelle Konfiguration des Frameworks bestimmt, welche die Datei standard.properties ausliest.

Übersicht der wichtigsten Klassen

MainApp

Die Klasse MainApp stellt die Hauptanwendung dar. Sie generiert den Szenengraphen und erzeugt eine oder mehrere ModellStructures, z.B. eine Helix. MainApp ist von Java3DApplet abgeleitet.

DataBackendManager

Der DataBackendManager ist ein Singleton, das dazu dient, die Kommunikation mit Jena zu übernehmen.

SceneManager

Der SceneManager ist ebenfalls als Singleton realisiert. Er bietet eine Schnittstelle, um Informationen über die Szene zu erhalten und Komponenten zu registrieren, die von überall aus erreichbar sein sollen.

EntityRepresentation

Eine EntityRepresentation ist eine Java-Bean ähnliche Klasse, die alle Informationen zu einem Entity kapselt bzw. auf Anfrage lädt, wenn diese noch nicht von der Ontologie angefordert wurden.

EntityVisualization

Die Darstellung eines Entities. Sie verwaltet die graphische Abbildung einer EntityRepresentation und hat eine Position im 3D-Raum.

EntityPainter

Ein EntityPainter ist der ModellStructure zugeordnet und kann ausgetauscht werden. Er übernimmt das Konstruieren der EntityVisualizations.

LoadUnloadByDistance Behavior

Das Java3D Behavior wird von Listenern benachrichtigt und veranlasst je nach Entfernung des Betrachters das Laden und Entfernen von Objekten.

KeyHelixBehavior

Das KeyHelixBehavior dient der Navigation per Tastatur und nimmt auch Shortcuts für Standard-Positionen entgegen. Das Tastatur-Verhalten ist der ModellStructure zugeordnet.

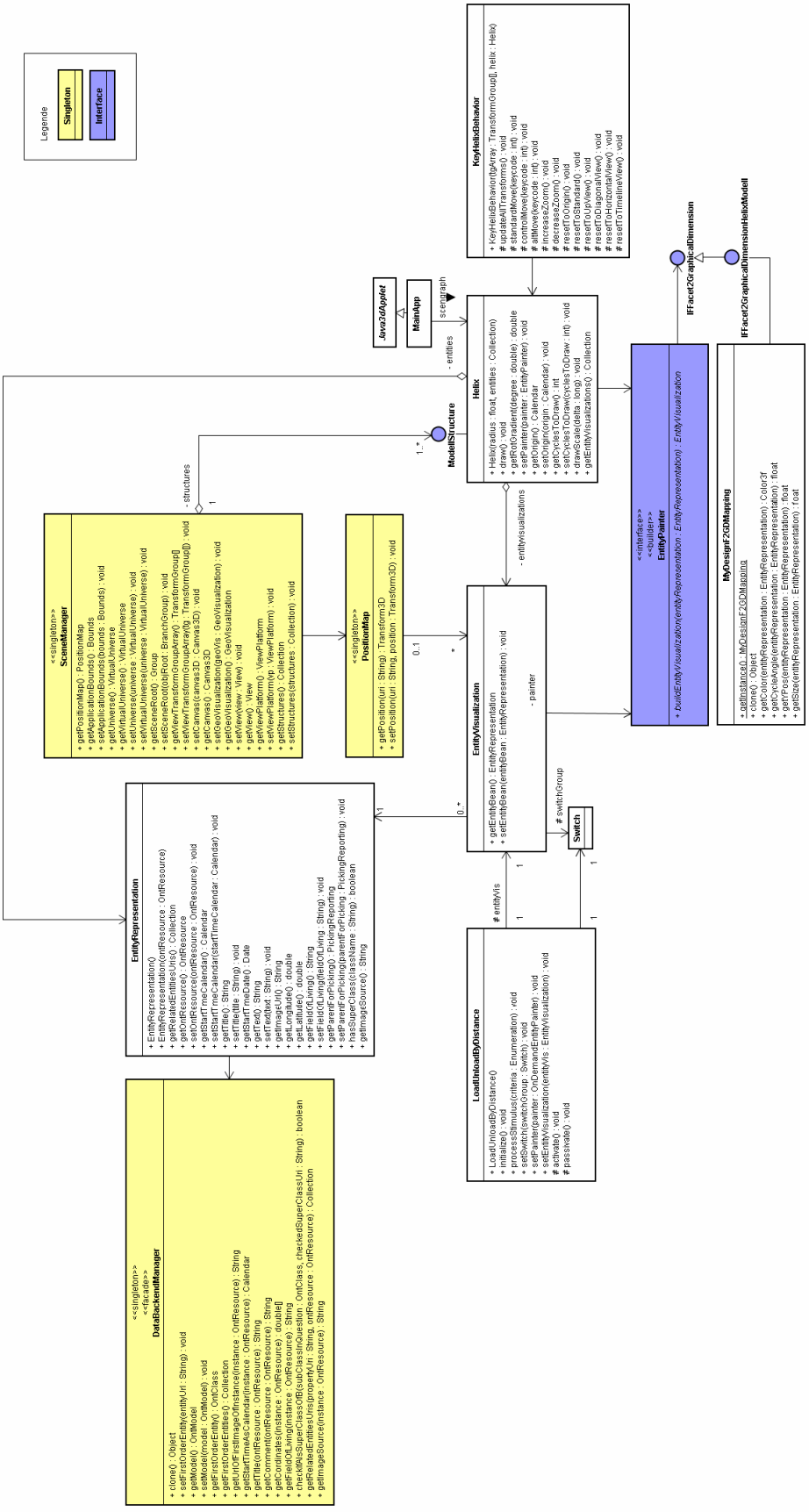


Abbildung 18 – Klassendiagramm: Übersicht der wichtigsten Klassen

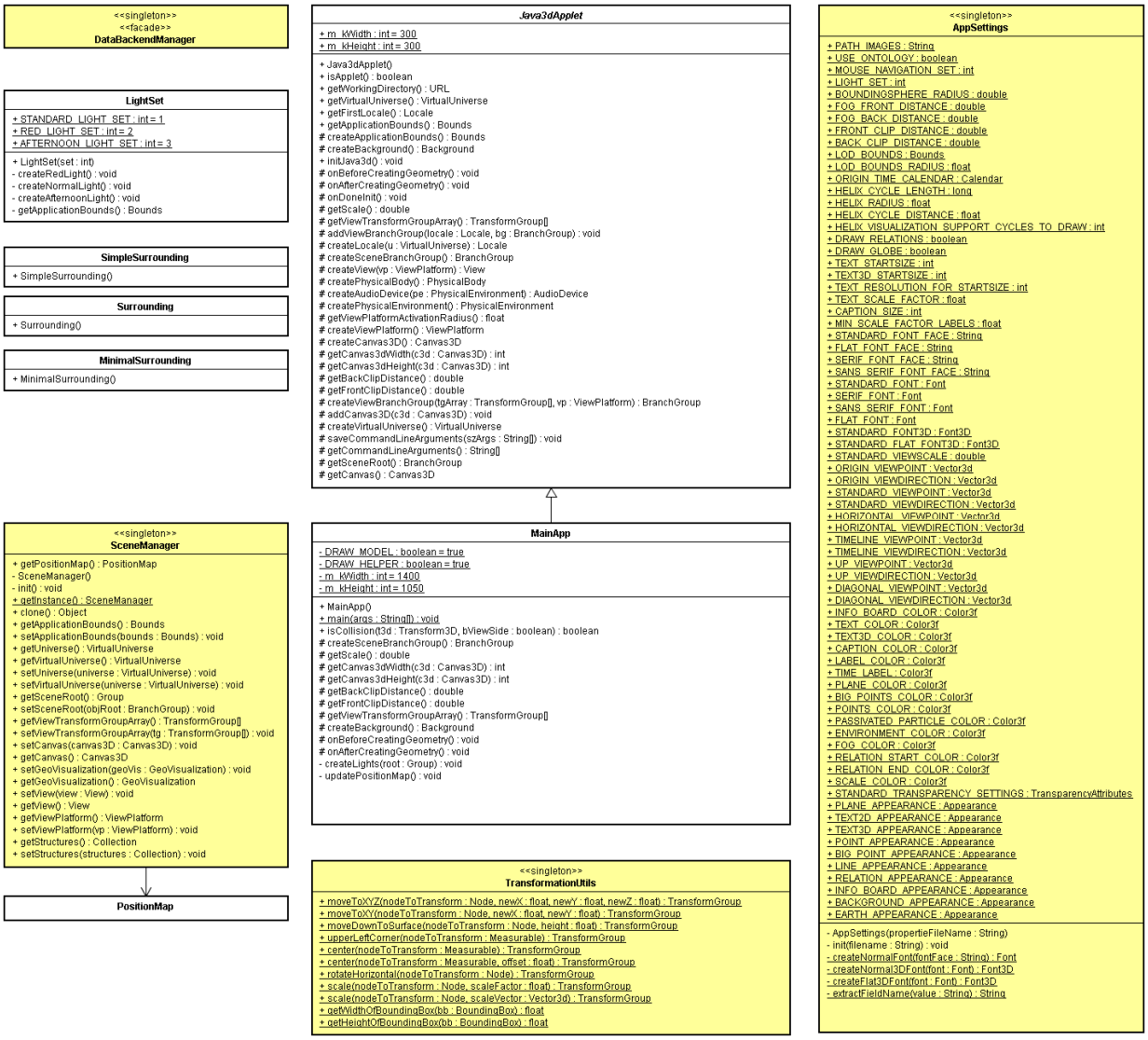


Abbildung 19 - Detailliertes Klassendiagramm ausgewählter Klassen

Packages

atomicelements
häufige genutzte Primitive, die optisch nicht weiter zerlegbar sind

visualizationsupport
Partikel, die als Hilfselemente dienen

structures
zusammengesetzte graphische Objekte

structures.painters
austauschbare, strukturgenerierende Klassen

behaviors
Aktionen, die auf dem Szenengraphen ausgeführt werden, und von Listenern aufgerufen werden

common
Einstellungs- und Helfer-Klassen

tests
Java and Jena Testprogramme

universe
Hauptanwendungen

scene
Sets von Dingen, die das Aussehen und die Navigation in der Szene bestimmen

Verzeichnisse

root properties-Dateien zur Konfiguration, bat-Dateien zum Starten der Anwendung unter Windows

ressources

images	Bilddateien, die in der Anwendung benötigt werden
images_lr	Bilddateien in verringerter Auflösung
ontologies	OWL-Dateien

doc

javadoc	
ontdoc	HTML-Dokumentation der Ontologien
uml	Diagramme zur Beschreibung des Frameworks im JUDE-Format und als bmp

etc ont-policy.rdf zur Konfiguration von Jena

5.3 Konfiguration

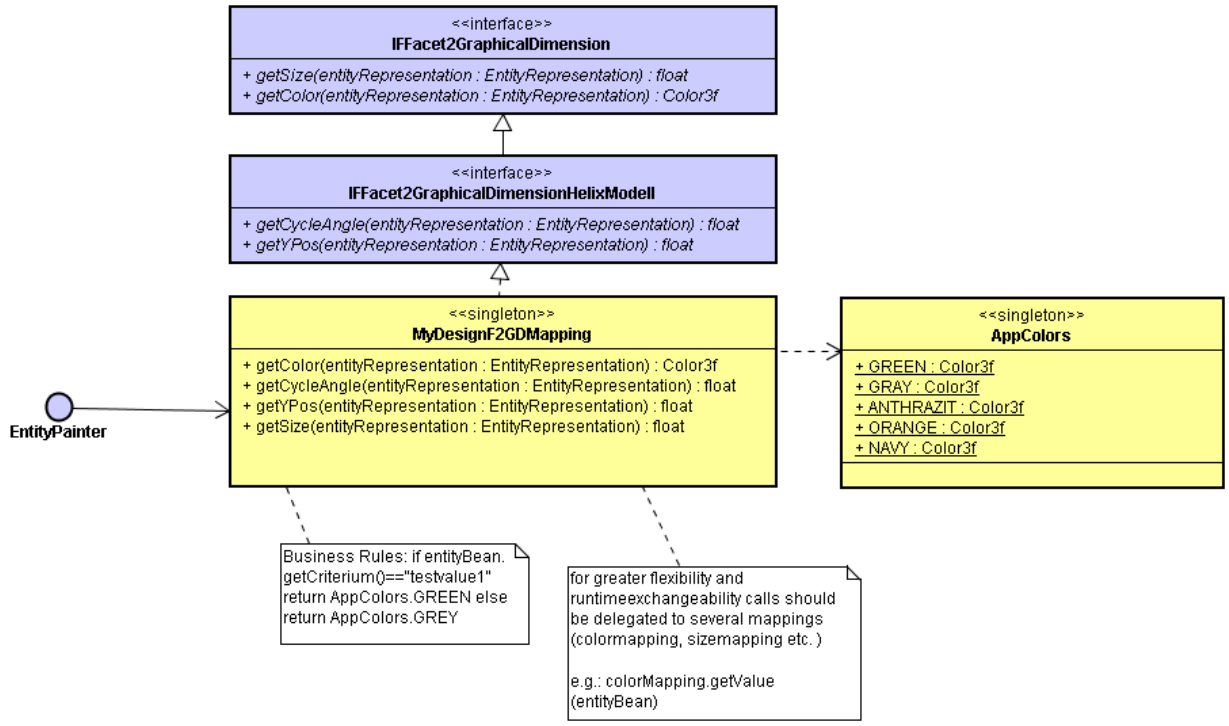
Die Konfiguration des Frameworks geschieht über die Singleton-Klasse common.AppSettings. Alle Klassen im Framework fragen hier ihre Standardwerte ab.
Die eigentlichen Einstellungen sind in einer properties-Datei (standard.properties) im Root-Verzeichnis gespeichert, welche von AppSettings bei der Initialisierung ausgelesen wird. Per Parameter kann beim Start der Anwendung eine andere properties-Datei angegeben werden (siehe Seite 82, Installation). Auf diese Weise können leicht mehrere Konfigurationen entwickelt, verglichen und gesichert werden.

5.4 DataBackendManager

Die Singleton Klasse verwaltet den zentralen Zugriff auf die Ontologien über die Jena-API. Sie ist verantwortlich für das anfängliche Erzeugen des Ontologie-Models mit Jena und später für die Vereinfachung von Anfragen an das Model. Über getModel() lässt sich jedoch auch jederzeit das Ontologie-Modell für komplexere Anfragen erhalten. Mehrfach genutzte Anfragen sollten im DataBackendManager formuliert und wiederverwertet werden.
[+] Durch Einführung eines Interfaces ließen sich mehrere verschiedene DataBackendManager zur Unterstützung verschiedener Backends alternativ verwenden, beispielsweise um eine Datenbank oder einen Webservice abzufragen. Die Methoden des DataBackendManager müssten dann jedoch, statt OntResource, einen String entgegen nehmen.

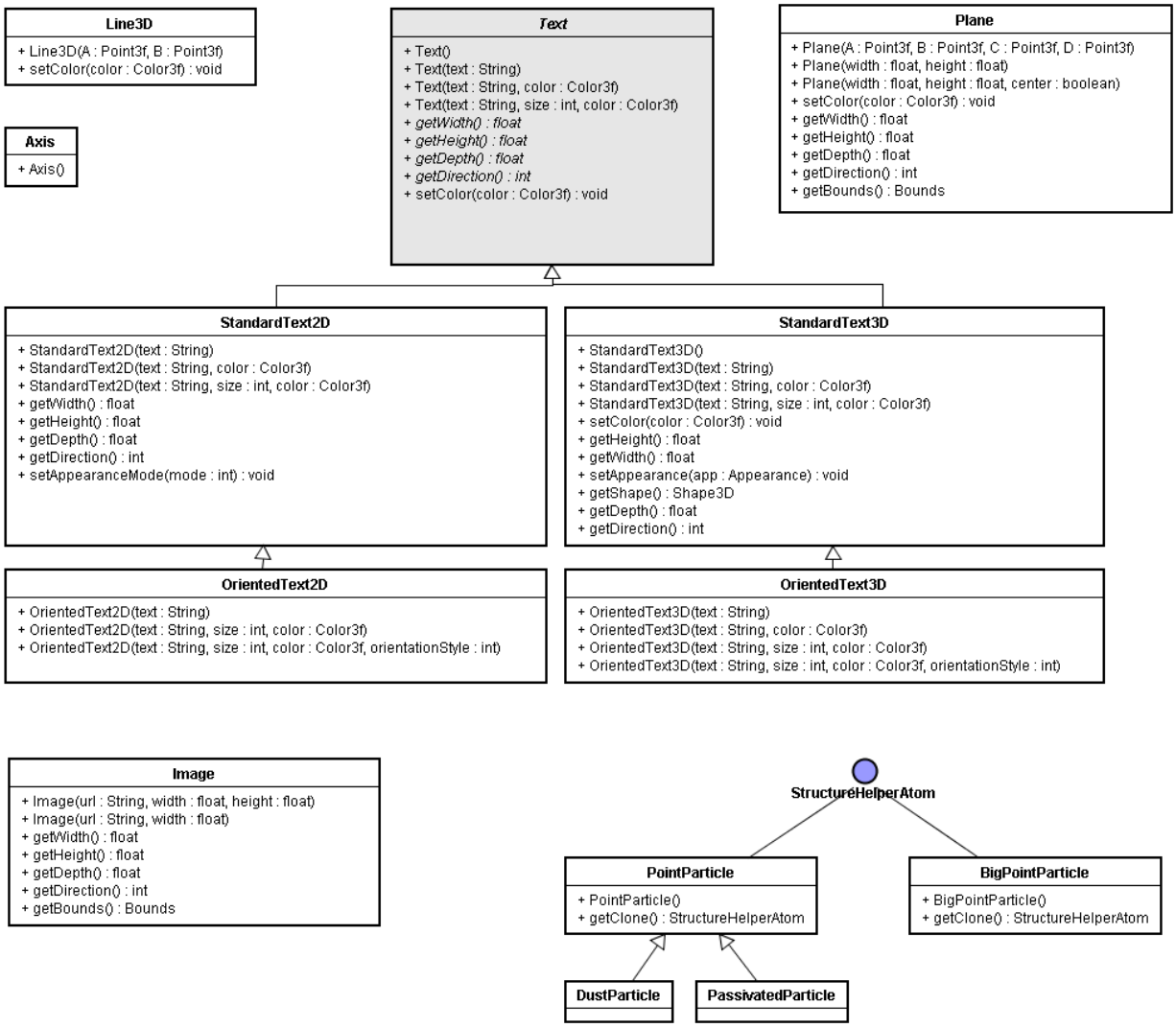
5.5 Mapping im Framework

EntityPainter stellt eine Anfrage an eine ihm zugeordnete Implementation des Interfaces IFFacet2GraphicalDimension, um die gewünschten Eigenschaften der ModellStructure zu erhalten. Für jede graphische Eigenschaft wird hier eine Methode definiert, welche die ihr übergebene EntityRepresentation abfragt und den graphischen Parameter zurückgibt. IFFacet2GraphicalDimension ist damit die bisherige, vereinfachte Umsetzung des in Abschnitt 4.4 beschriebenen Prinzips mit fixer Abbildung durch den Quellcode.
[+] Für mehr Flexibilität sollte jede graphische Eigenschaft ein eigenes Mapping erhalten. Diese könnten dann zu einem Aggregat zusammengesetzt und Methodenaufrufe an die verschiedenen Mappings delegiert werden. Einzelne Mappings könnten so auch zur Laufzeit ausgetauscht werden.



5.6 atomicelements

In diesem Package befinden sich häufige genutzte Primitive, die optisch nicht weiter zerlegbar sind.



Text, StandardText2D, StandardText3D, OrientedText2D, OrientedText3D

sind Textobjekte, die von BranchGroup erben und direkt im Szenengraphen verwendet werden können. StandardText2D verwendet Texturen, um Schrift darzustellen. StandardText3D generiert echten 3D-Text und verbraucht vor allem bei Serifenschriften viel Speicher.
OrientedText2D und OrientedText3D verwenden ein OrientedShape3D und lassen sich zum Benutzer rotieren (Billboard Behavior).

Image

zeichnet ein Bild. Das Bild wird vom Mittelpunkt in Richtung Süd-Ost gezeichnet.

Plane

ist ein Rechteck, welches zentriert oder in Richtung Süd-Ost gezeichnet werden kann.

Line3D

zeichnet eine Linie, die 2 Punkte im 3D-Raum verbindet und z.B. zur Darstellung von Relationen genutzt wird. (Siehe auch Abschnitt über Relationen).

VisualizationHelperAtom, PointParticle, BigPointParticle, PassivatedParticle

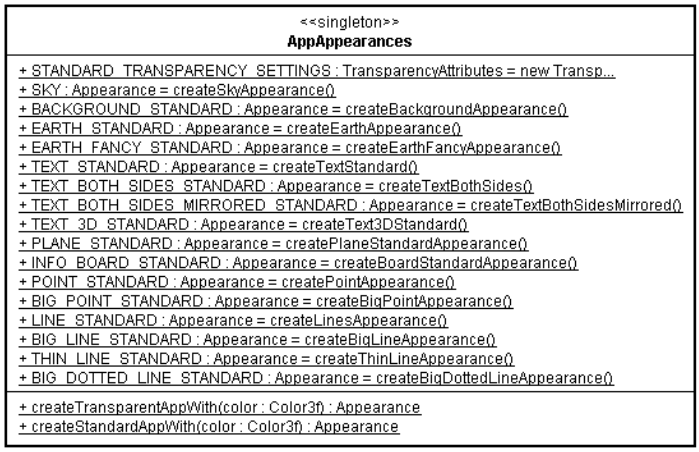
sind annähernd punktförmige Elemente, welche die Visualisierung von Strukturen unterstützen können und auch in den Skalen verwendet werden.

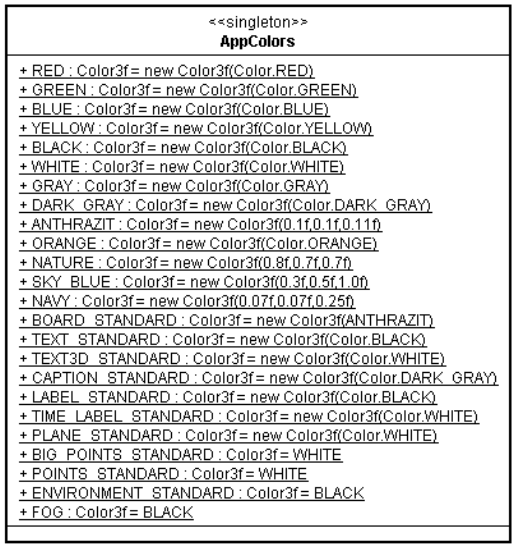
Axis

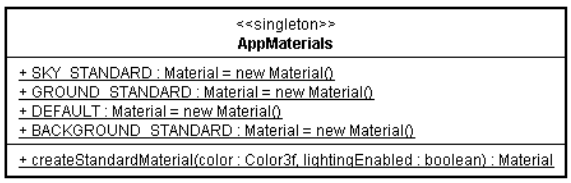
zeichnet eine Achse zur Kennzeichnung eines Koordinatensystems.

5.7 Appearance Bibliothek

Die Klassen AppAppearances, AppMaterials und AppColors dienen als Bibliothek für häufig verwendete Appearances, Materials und Colors. Über statische Methoden lassen sich außerdem Standard-Appearances und -Materials erhalten. Durch die zentrale Erzeugung der Appearances lassen sich diese einfach wieder verwenden.

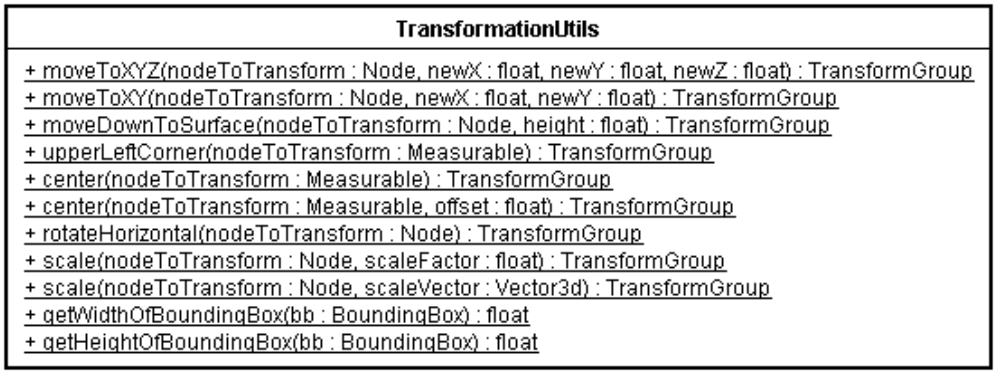






5.8 TransformationUtils

Bei den TransformationUtils handelt es sich um eine Hilfsklasse, die Transformationen auf übergebenen Objekten ausführt und diese gekapselt in TransformGroups zurückgibt. Alle Methoden sind statisch und können so einfach genutzt werden.



5.9 Structures

Das Package structures beinhaltet alle strukturerzeugenden Klassen. Das sind Modelle, EntityPainter (im Unter-Package structures.painters), EntityVisualizations und InfoBoards, auf die in den nächsten Abschnitten noch näher eingegangen wird.
Das Interface ModellStructure legt fest, welche Bedingungen eine Klasse erfüllen muss, die als Modell im Framework genutzt werden soll.

Helix-Modell

Helix (dtsch. Wendel) ist eine Implementation des Interface ModellStructure und ermöglicht die wendelförmige Anordnung von Entities. Die Form der Helix bringt eine Verdichtung der sequentiellen Information mit sich (wie auch bei der DNS-Doppelhelix in der Biologie). Im Vergleich zu einem Zeitstrahl liegen mehr Ereignisse im gleichen Volumen. Die dritte Dimension wird durch die Aufwicklung zur Helix genutzt. Zusätzlich hat die resultierende Grundform des Zylinders den Vorteil, per Kopfdrehung eine große Anzahl von Elementen sichten zu können. So ergibt sich ein Panorama. Reicht dies nicht aus, kann die Drehbewegung mit der simplen Auf-und-Ab-Bewegung entlang des Zeitstrahls bzw. der Y-Achse überlagert werden. Auf diese Weise verbindet die Geometrie der Helix das Vorwärtskommen mit der Bewegung in Zyklen
Elemente in der Helix sollten so beschaffen sein, dass sie von der Rückseite betrachtet, aus größerer Entfernung zwar sichtbar sind, bei geringer Entfernung jedoch transparent, um die Sicht auf gerade fokussierte Elemente nicht zu stören. Der Blick auf das gerade fokussierte Element sollte durch vollständiges Ausblenden störender Elemente gesichert sein.

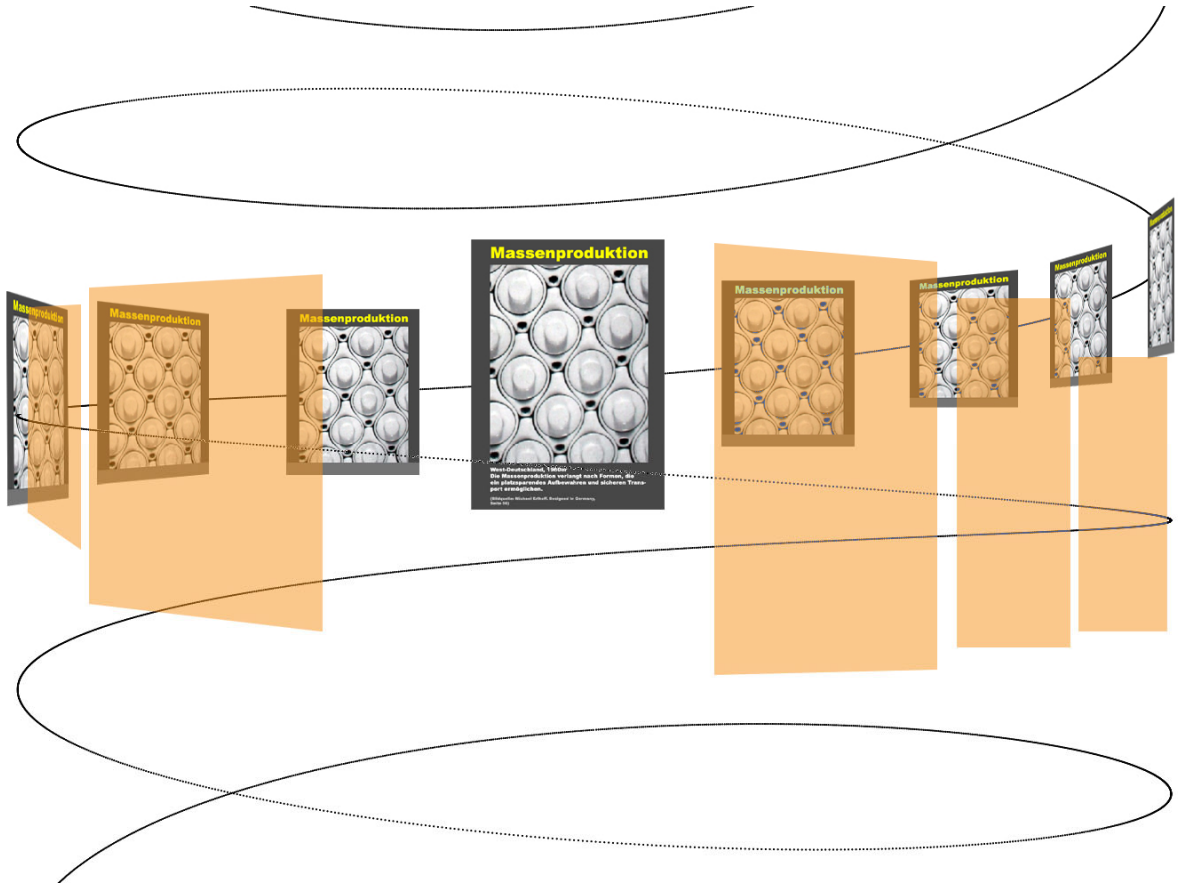
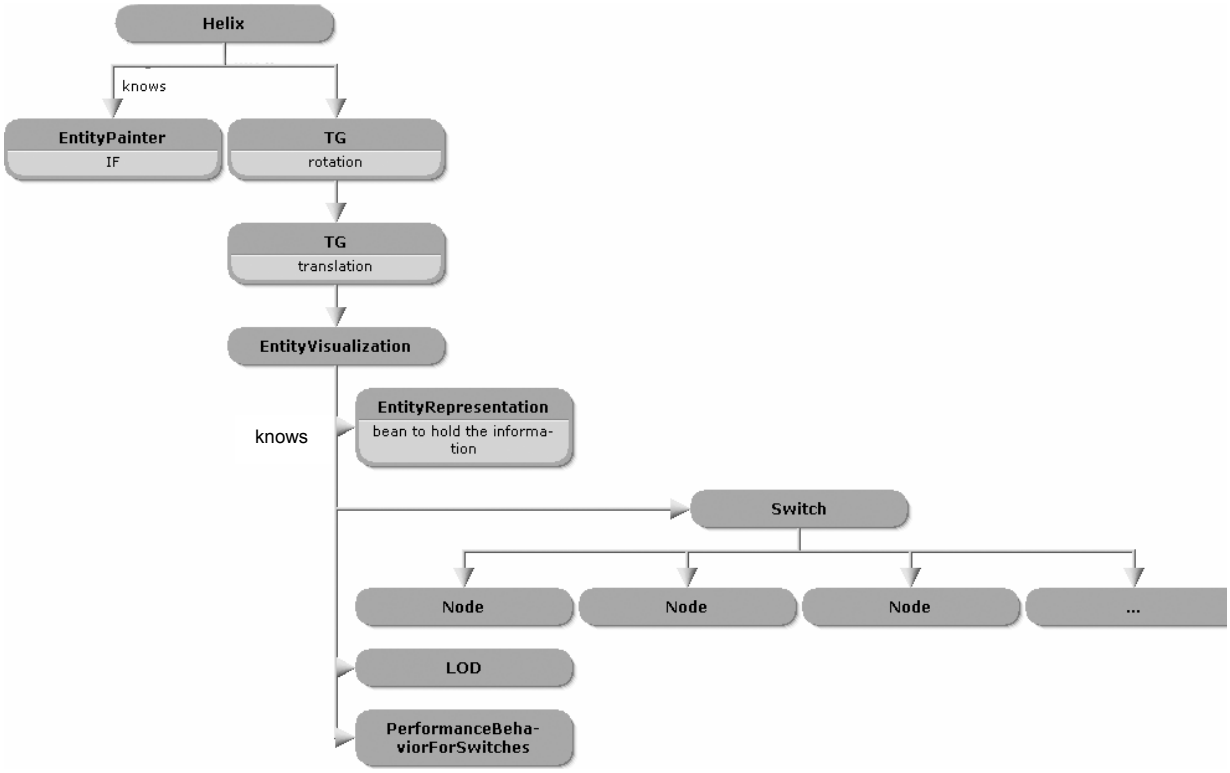


Abbildung 20 – Helix-Modell mit teils ausgeblendeten, teils transparent und eintönig dargestellten InfoBoards

Szenengraph der Helix

Stellvertretend für alle EntityVisualizations wurde hier der Szenengraph für die Positionierung einer EntityVisualization gezeichnet. Alle unbeschrifteten Pfeile stellen die child-Relation des Szenengraphen dar.



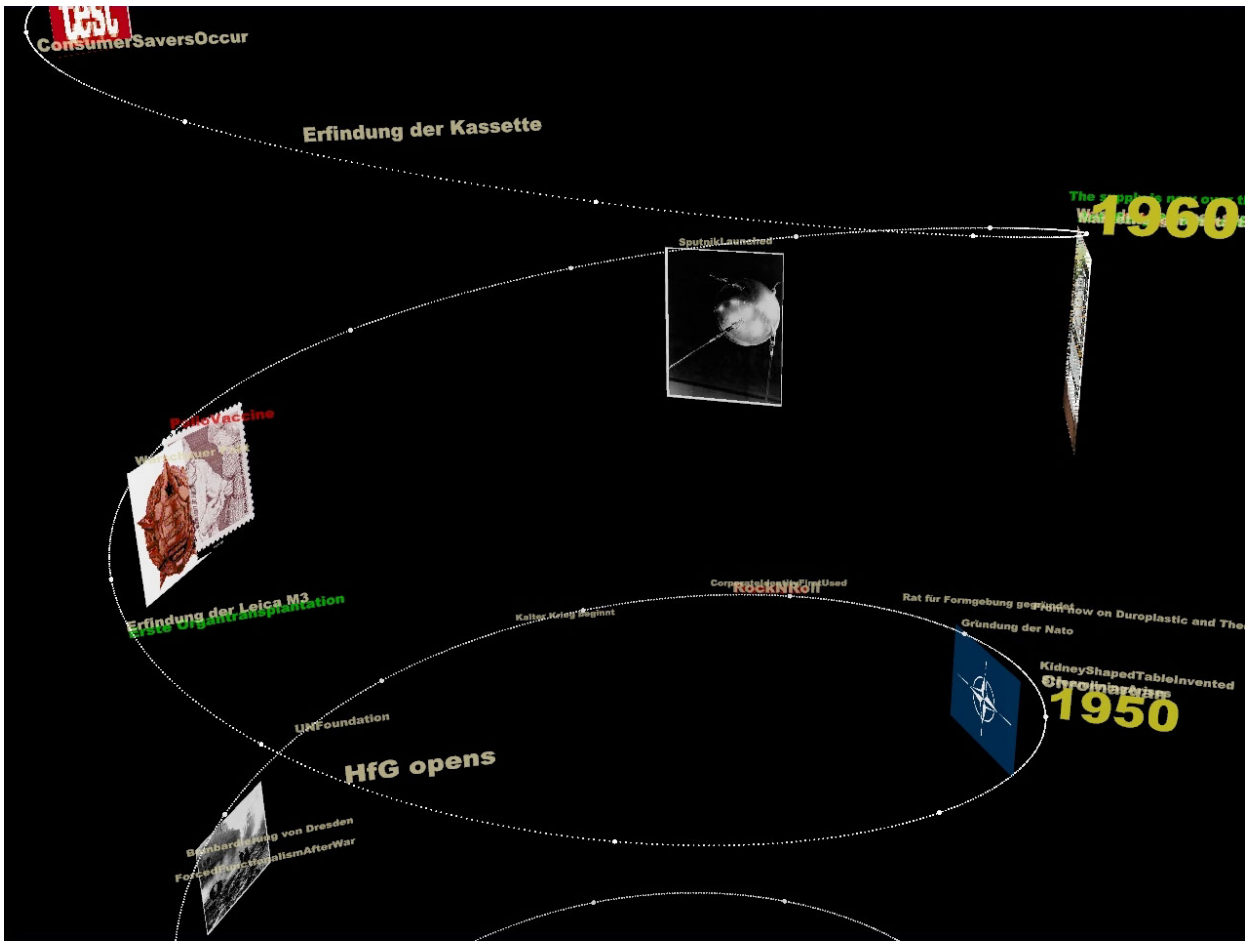
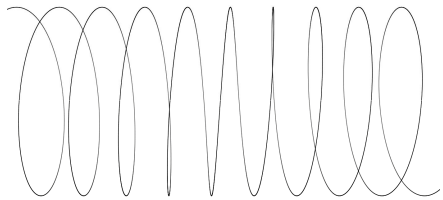


Abbildung 12 – Screenshot: Anwendung der Zeit-Helix

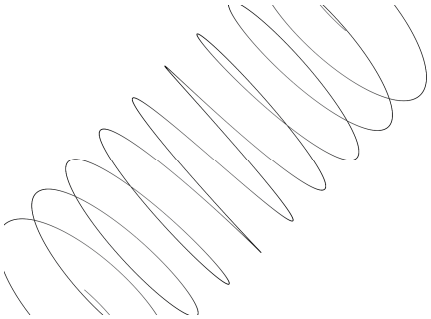
Das Modell bietet mehrere Ansichtsmöglichkeiten, die verschieden interpretiert werden können:



Zeitstrahl horizontal

Der fortschreitende Anteil der zyklischen Bewegung wird sichtbar und entspricht der Leserichtung von links nach rechts.

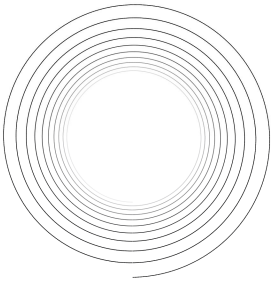
- (Ansteuerung `Strg+H`)



Zeitstrahl als Graph

Die Ausrichtung der Form, die sich durch diese Sicht ergibt, entspricht ebenfalls der Leserichtung. Ein Verlauf von links oben nach rechts unten wurde jedoch verworfen, da die Form in der dargestellten Art und Weise eher an einen Graphen in einem Diagramm erinnert, der mit der Zeit steigt.

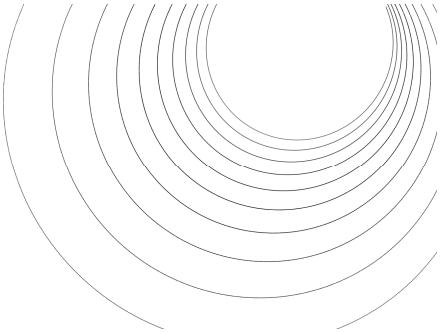
- (Ansteuerung per `Strg+T`)



Sicht nach unten – Jahresringe

Die zum Zeitverlauf orthogonale Ansicht hat Diagrammcharakter und bietet ein flächiges Erscheinungsbild.

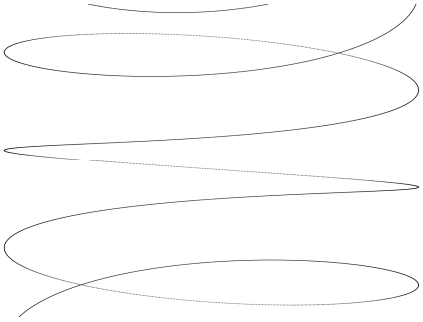
- (Ansteuerung per `Strg+U`)



Diagonale

Die im Gegensatz zur orthogonalen Sicht leicht verschobene Perspektive bringt mehr Tiefe in die Darstellung. Bei Einsatz von Billboard Behavior wirkt die Ansicht dennoch flächig.

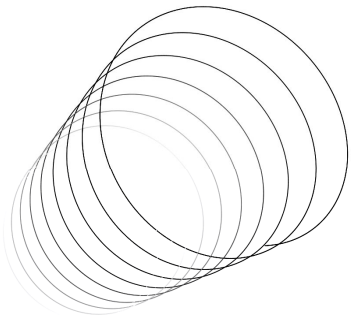
- (Ansteuerung per `Strg+D`)



Normalansicht

Die Standardperspektive lässt den Blick auf Details zu und zeigt einen Zyklus oberhalb und unterhalb als Kontext. Bei Navigation mit der Tastatur entlang der Struktur scheinen sich die Entities entlang der Struktur zu verschieben[2]. Es kann beobachtet werden, wie sie sich langsam nähern, bis sie dann im Zentrum detailliert dargestellt werden.

- (Ansteuerung per `Strg+N`)



Übersicht

Bei der freien Navigation lässt sich zu einer weit entfernten Übersichtsperspektive wechseln. Ein Eindruck zeitlicher Dimension ist in dieser Ansicht am besten ablesbar. Daneben kann sie zur Orientierung beitragen. Diese Ansicht kann bei eingeschränkter Navigation auch im HUD zusätzlich angezeigt werden.

Erstellen der Helix in der MainApp

Das folgende Kollaborations-Diagramm veranschaulicht die Vorgänge bei der Erstellung der ModellStructure Helix. Schritt 1 und 2 dienen der Erstellung von Entities durch den DataBackendManager.

[2] Der Effekt der relativen Bewegung wird im Abschnitt Gestaltung des Mediums auf Seite 74ff. beschrieben.

Die erstellten Entities werden dann zur Erzeugung einer neuen Helix verwendet. Zum Zeichnen der EntityVisualizations wird die Helix mit einem austauschbaren EntityPainter ausgestattet. Die Methode draw() veranlasst die ModellStructure sich zu zeichnen, drawScale(int ms) ist optional aufzurufen und zeichnet eine Skala (siehe auch Abschnitt zu Skalen weiter unten). Abhängig von der angegeben Distanz in ms wird eine unterschiedliche Visualisierung jeder Markierung der Skala gewählt.
Jede ModellStructure wird zu der Collection structures im SceneManager hinzugefügt und kann über getStructures() wieder abgefragt werden. Die Hauptanwendung ist nach Erstellung der Helix, wenn die Struktur dem Szenengraphen hinzugefügt wurde und die Positionen der einzelnen EntityVisualizations feststehen, dafür verantwortlich jede dieser Positionen in der PositionMap zu vermerken.

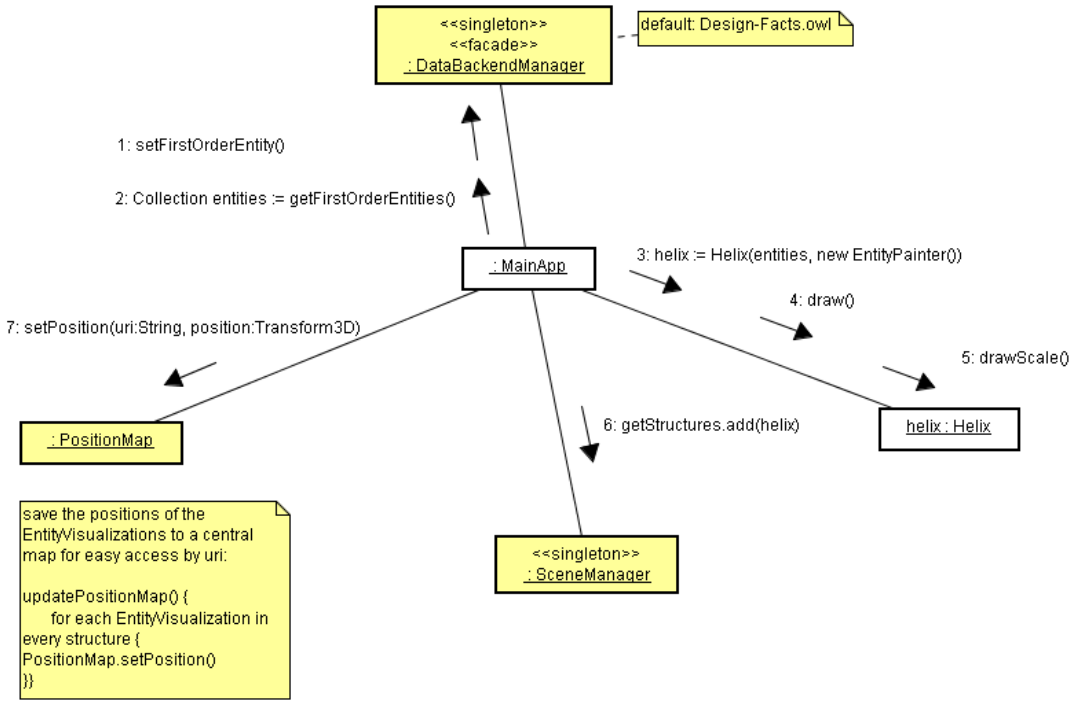
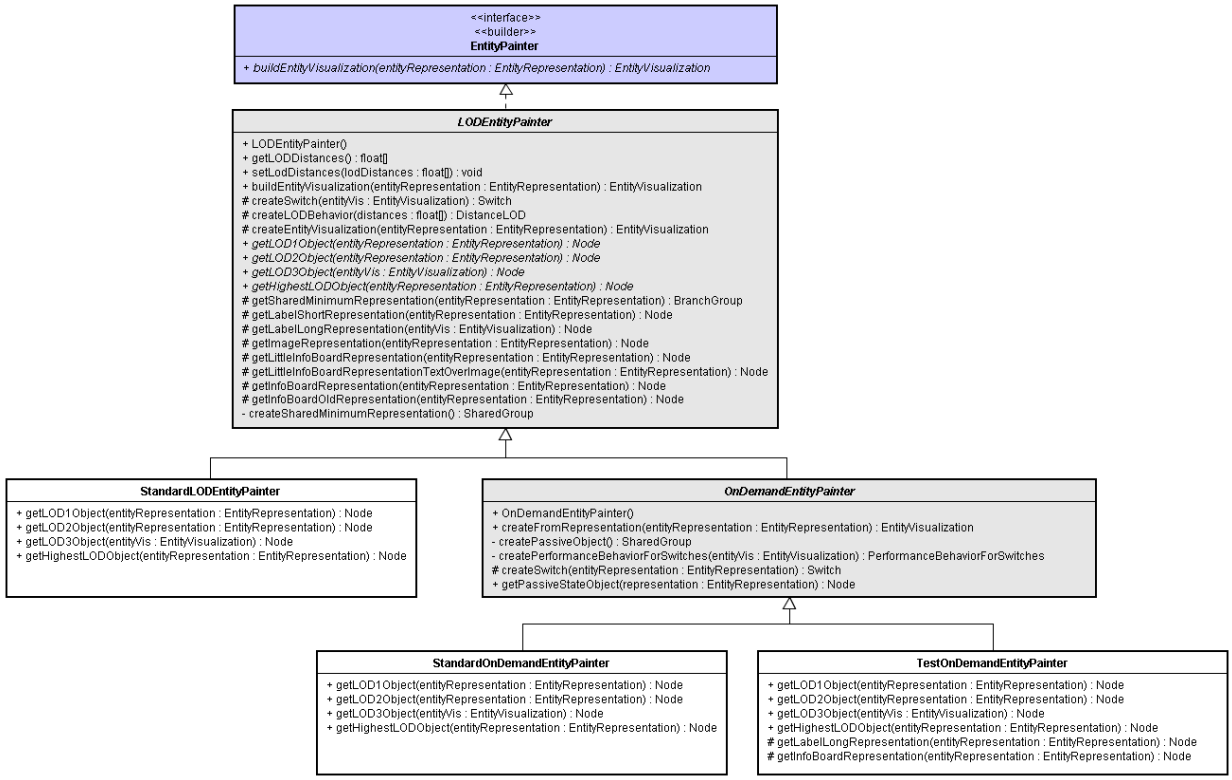


Abbildung 21 – Erstellen der Helix

EntityPainter

EntityPainter ist ein Interface, das austauschbare Objekte beschreibt, die genutzt werden um, EntityVisualizations aufzubauen und ggf. verschiedene Levels of Detail zu erzeugen. Das Design Pattern Builder wird eingesetzt [GAMMA_ET_AL 1996]: EntityPainter spielt die Rolle des Builders, EntityVisualization übernimmt die Rolle des Directors und der aus verschiedenen Nodes, den LOD, zusammengebaute Switch stellt das aufgebaute Produkt dar. Über die ModellStructure konfiguriert die MainApp in der Rolle des Clients den Direktor.
Neue EntityPainter können von den abstrakten Klassen LODEntityPainter und OnDemandEntityPainter abgeleitet werden, in dem die Methoden getLOD<X>Object(EntityRepresentation entityRepresentation) implementiert werden und darin ein LOD zugeordnet wird. Zudem ist der Konstruktor von LODEntityPainter eine Template Method, die getLODDistances() aufruft, um die Abstände für den Übergang von einem LOD in ein anderes zu bestimmen. Durch Überschreiben dieser Methode kann eine neue Unterteilung der LOD-Sichtbarkeiten erreicht werden.



Erzeugung einer EntityVisualization mit LOD-Behavior

Durch die Helix wird der LODEntityPainter beauftragt, eine EntityVisualization zu bauen. Dazu erstellt er diese, erzeugt einen Switch und ruft seine Methoden auf, um die verschiedenen Level für das LOD-Behavior zu generieren. Anschließend wird ein LOD-Behavior erzeugt und der Switch bei diesem angemeldet. Beide Objekte werden dem Szenengraphen hinzugefügt, indem sie der EntityVisualization zugeordnet werden.

Im Fall eines OnDemandEntityPainter sollten anfangs alle Child-Objekte des Switch mit Platzhaltern gefüllt werden, welche dann später bei der Aktivierung ersetzt werden. Um bei der Aktivierung die echten LOD erzeugen zu können, wird dem verantwortlichen Behavior der Painter mit übergeben.

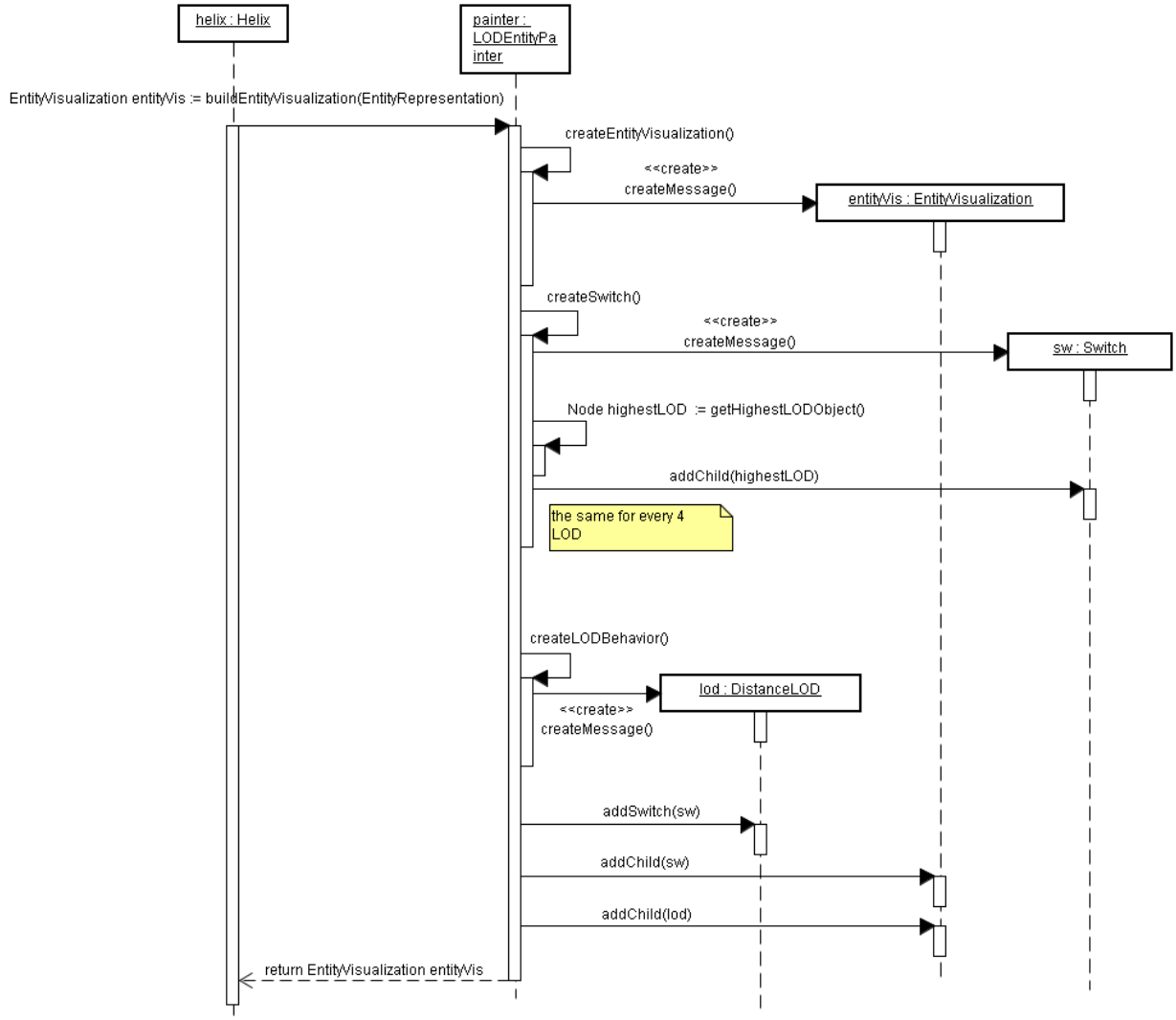
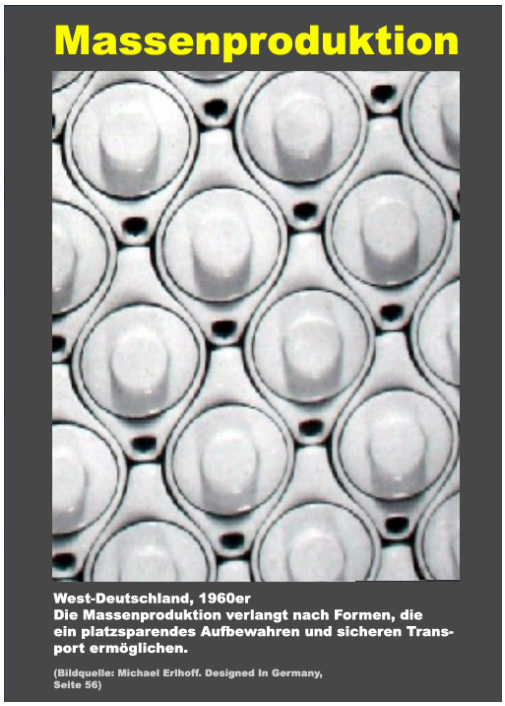


Abbildung 22 – Sequenz Diagram: Zeichnen einer einzelnen EntityVisualization in der Helix

InfoBoards

InfoBoards sind Strukturelemente, die Zusammenstellungen von Text und Bild zu einer flächigen zweidimensionalen Einheit darstellen. Diese können von EntityPaintern genutzt werden, um hohe, komplexere LOD darzustellen.



Skalen

Im Gegensatz zur relativen Einordnung durch die Beziehung zu anderen Elementen, z.B. der relativen Lage eines Ereignisses zwischen zwei anderen Ereignissen, ermöglichen Skalen ein absolutes Ablesen eines Kriteriums.
Ein Beispiel dazu: „Ereignis A fand zwischen 2000 und 2005 statt“ anstelle von „Ereignis A fand nach Ereignis B statt“.
Für manche graphische Eigenschaften wie räumliche Dimensionen, können Skalen direkt in den 3D-Raum integriert und mit Struktur-Hilfselementen gekoppelt werden oder diese ersetzen. So kann z.B. für die Struktur-Hilfselemente der Zeit-Helix gelten: 1 Punkt = 1 Woche / Monat. Dadurch, dass die Skala die Daten überdeckt oder zumindest davon ablenkt, können neue Probleme entstehen.
Für andere graphische Eigenschaften, wie etwa die Farbe, kommen Legenden in Frage. Diese Legenden können losgelöst vom 3D-Raum im HUD Display liegen. Generell sollte aber versucht werden, ein intuitives Verständnis der Abbildung von graphischen Eigenschaften zu erzielen. Legenden sollten also möglichst nur zusätzlich angeboten werden.

5.10 LOD

Level Of Detail Mechanismen, also die Anzeige von verschieden detaillierten Repräsentationen eines Gegenstandes je nach Ausprägung eines Parameters, in diesem Fall der Nähe zum Betrachter, sind ein zentraler Bestandteil der Anwendung. Einerseits sind sie Vorrausetzung für Verfahren, um Speicher zu sparen, andererseits lässt sich so eine Informationsüberflutung und Ablenkung vom fokussierten Objekt verhindern.

Bei der Auswahl der verschiedenen Ebenen ist es sinnvoll, die Position und Größe von einem Element zum anderen möglichst beizubehalten, um Änderungen in der Bildstruktur überschaubar zu halten und einen Zusammenhang zwischen den einzelnen Repräsentationen zu bekräftigen. Die folgenden Abbildungen stellen die verschiedenen LOD dar, welche die Klasse LODEntityPainter liefert (abgesehen von der Partikel-Darstellung): MinimumRepresentation, LabelShort, LabelLong, Image und InfoBoard. Die Partikel-Darstellung kann genutzt werden um darzustellen, dass überhaupt etwas an einer gewissen Position vorhanden ist. Die Farbe der Überschrift und der Partikel sowie der Punkt-Darstellung kann über alle LOD hinweg einheitlich verwendet werden, wenn diese eine Eigenschaft des Entites abbildet.
Die minimale Distanz zum Betrachter für den Übergang zu einem neuen LOD wird über ein Array von Distanzen geregelt. Das genaue Einstellen dieser Distanzen entscheidet über die Gebrauchstauglichkeit.
Im Framework wird LOD über einen Java3D Switch und ein Java3D DistanceLOD-Behavior erzeugt.

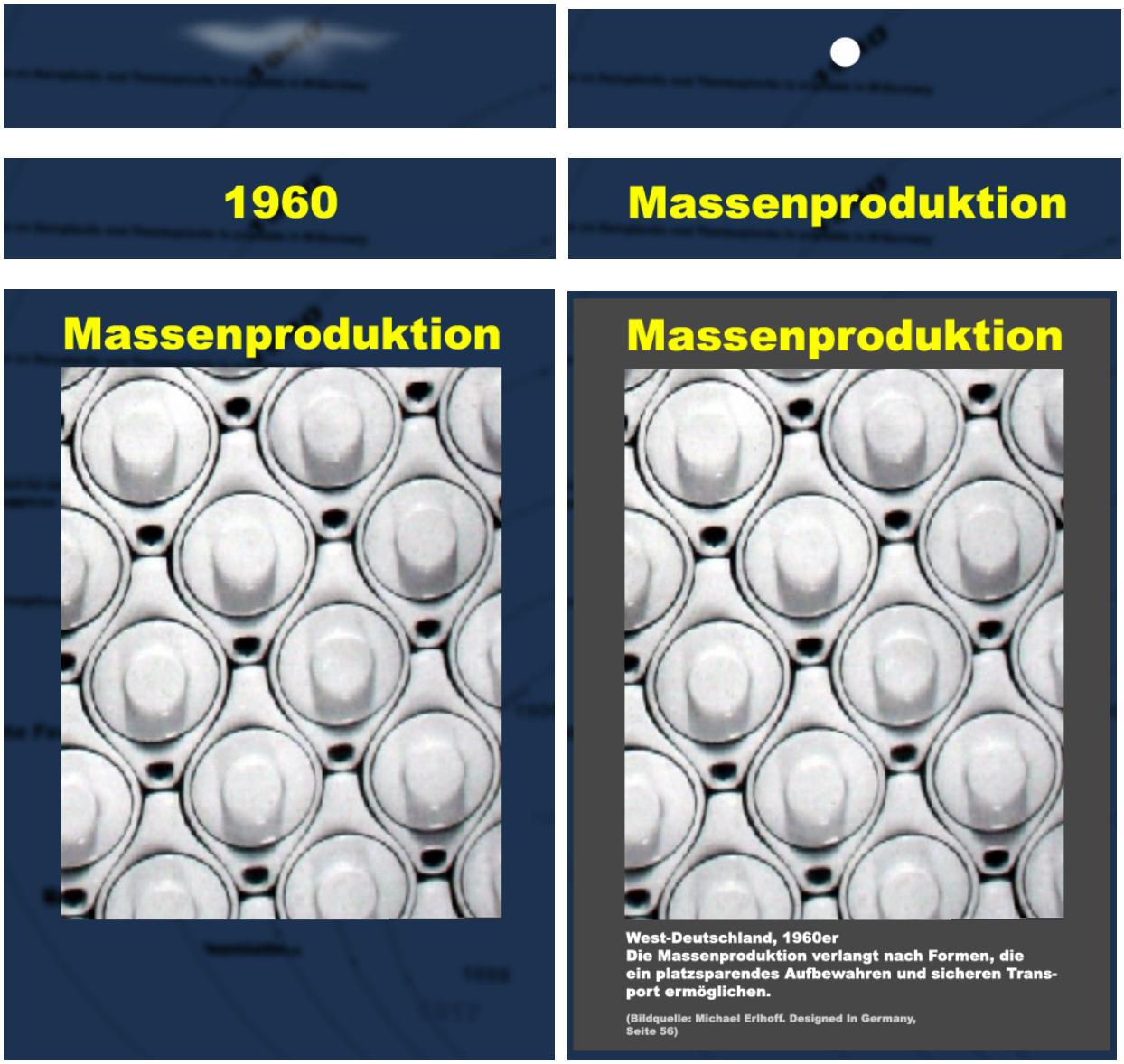
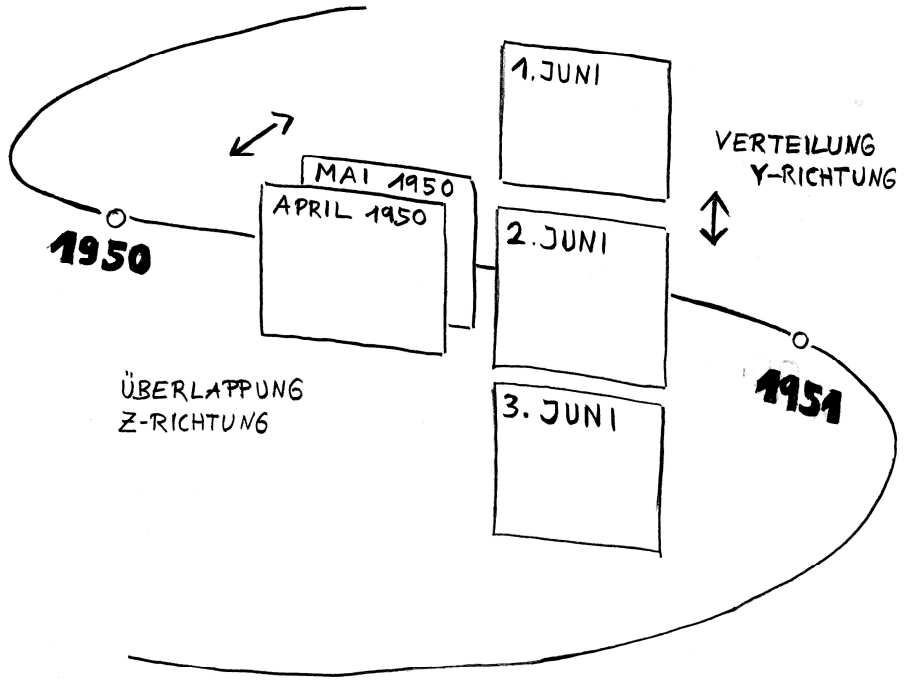


Abbildung 23 - Verschiedene Levels of Detail

5.11 Häufung von Einträgen [+]

Dieses Problem betrifft beispielsweise die Anordnung von Ereignissen im Raum, wenn nach der Zeit angeordnet wird und sehr viele zeitlich eng beieinander liegende oder im Extremfall gleichzeitige Ereignisse existieren. Das Problem hängt stark von der gewählten Skalierung ab. Bei einer sehr groben zeitlichen Darstellung, wie etwa ein Jahrhundert pro Zyklus im (Zeit)Helix-Modell, ergeben sich schneller Überlappungen als bei einer Jahresdarstellung. Die Detailstufe, der Zoomfaktor und der Radius der Helix sind weitere Parameter der Problematik. Zur Auflösung gibt es verschiedene Mechanismen, keiner davon wurde bisher in die Beispielimplementierung aufgenommen. Grundsätzlich muss überlegt werden, ob die Strategie im Voraus angewendet wird, oder erst nach Bedarf, wenn es zu Konflikten kommt.

Mechanismen zum Lösen von Überlagerungen



Graphischer Ansatz:

- Ein eleganter, jedoch nur bis zu einem gewissen Grad ausreichender Ansatz, ist die Änderung der Perspektive. Aus einem geschickten Winkel betrachtet, reduziert sich die gegenseitige Überdeckung der Elemente. Dies ist auch der Ansatz, den Kullberg [KULLBERG 1996] bei der Darstellung vieler Fotos auf engem Raum wählt.

Verteilen in X-Richtung

- Hierbei wird eine Schwächung der Bedeutung der graphischen Dimension in Kauf genommen, was jedoch durchaus vertretbar ist, wenn die Abweichung nicht zu groß wird. Die mathematische Genau-

igkeit kann in vielen Fällen der besseren Darstellung geopfert werden. Falls es sich um kritische Daten handelt, sollte diese Abweichung in jedem Falle deutlich sichtbar angemerkt werden.

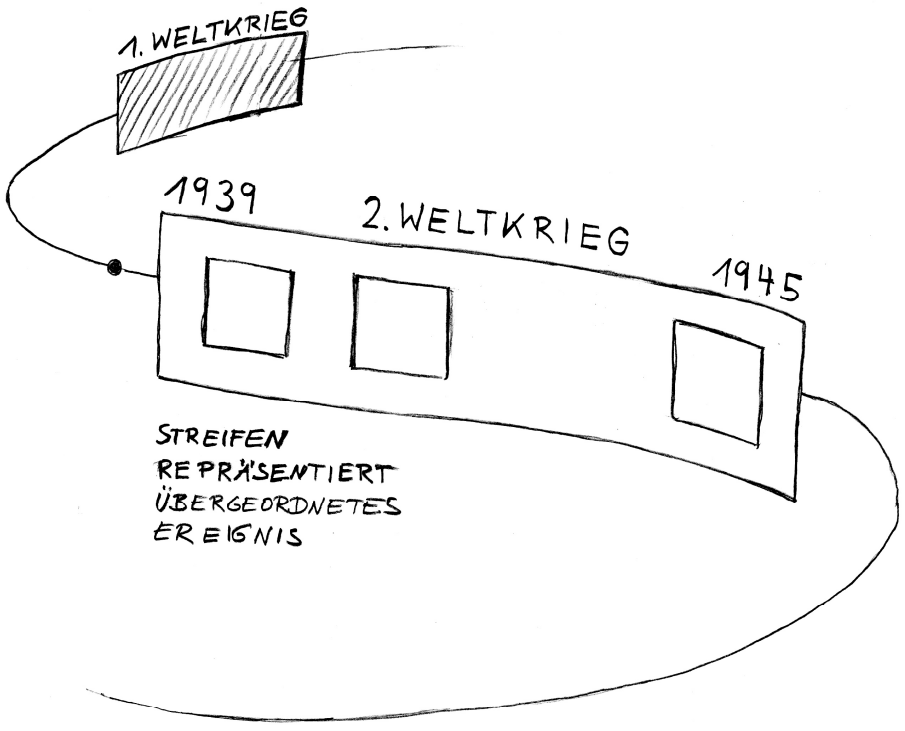
Ausweichen in eine andere graphische Dimension

- In Z-Richtung: Hierbei erfolgt die Auflösung der Überlagerung durch Registerkarten ähnliches Anordnen der konkurrierenden Ereignisse hintereinander. Durch Perspektive oder leichtes Verschieben muss gewährleistet werden, dass alle Elemente sichtbar sind.
- In Y-Richtung: Wenn gleichzeitige Ereignisse übereinander angeordnet werden, muss der Abstand zum nächsten Zyklus oberhalb und unterhalb ausreichend groß bleiben, deswegen kann dieser Mechanismus nur bis zu einer gewissen Anzahl von gleichzeitigen Ereignissen angewendet werdet.

KULLBERG lehnt die Möglichkeit in eine andere räumliche Dimension auszuweichen ab, da er die visuelle Struktur gefährdet sieht. In Maßen halte ich dieses Prinzip aber für vertretbar, wenn die Struktur selbst gefestigt genug ist.

Clusterbildung

- Eine weitere Möglichkeit besteht im Zusammenfassen von untergeordneten Ereignissen. Voraussetzung ist, dass eine entsprechende Hierarchie (partOf–Relation angelegt wurde). Wenn diese existiert, können untergeordnete Ereignisse vorerst ausgeblendet werden. Bei zunehmender Detaillierung und mehr Raum werden diese eingeblendet. Zwei Möglichkeiten bieten sich nun, mit dem übergeordneten Ereignis zu verfahren: Entweder wird es ersetzt, oder, wie in der Abbildung, als längere Phase über ein Band dargestellt, das die Kind-Ereignisse integriert. Der Ansatz löst jedoch nicht das Problem für Ereignisse, die verschiedenen Hauptereignissen angehören.



Dieser Ansatz kann mit der ebenfalls von KULLBERG beschriebenen Lösung für Verdichtung von Entities durch Interaktion kombiniert werden. Auf Verlangen, z.B. Mouse-Over, tritt das ausgewählte Entity

aus der Menge heraus und zeigt sich dem Betrachter. Wird es nicht mehr näher betrachtet, kehrt es in den ursprünglichen Kontext zurück und positioniert sich gemäß seinem Wert.

5.12 Darstellung von Relationen [+]

Grundsätzlich kann man unterscheiden zwischen gerichteten (asymmetrischen) und ungerichteten (symmetrischen) Relationen. In einigen Fällen ist außerdem noch die Darstellung von reflexiven Relationen interessant, also Beziehungen eines Elements zu sich selbst.
Typischerweise werden Verbindungslinien eingesetzt, um Relationen in graphischen Darstellungen darzustellen. Ein Pfeil anstelle einer Linie kann die Asymmetrie einer solchen Verbindung verdeutlichen.
Neben der einfachen Linie sind Bänder und auch animierte Verbindungen, z.B. Punkte, die sich entlang einer gedachten Linie bewegen, denkbar.
Anstelle von Pfeilen lassen sich alternativ Bänder mit Farbverlauf oder animierte, sich in eine Richtung bewegende, Elemente verwenden.
Diese Darstellungen fügen sich in manchen Fällen besser in das Gesamterscheinungsbild ein und können zur Beruhigung des Bildes beitragen. Hierbei darf nicht übermäßig Gebrauch von Animation gemacht werden. Denkbar ist die Darstellung der Relation nur auf Verlangen (z.B. bei Klick oder Mouse-Over).
Bei der Darstellung von transitiven Relationen muss entschieden werden, ob dies in der graphischen Darstellung Beachtung finden soll. Ein Kompromiss wäre die Darstellung bis zu einer gewissen Tiefe und anschließende langsame Ausblendung durch Transparenz.
Zur Unterscheidung verschiedener Relationstypen lässt sich wiederum das gesamte graphische Vokabular benutzen, wobei Orientierung und Länge jedoch vorgegeben sind. Außerdem empfehlen sich bei Linien fixe Linienstärken (z.B. Haarlinien), um zu garantieren, dass die Relation bei jeder Entfernung lesbar ist.
Auch eine explizite Kennzeichnung per Beschriftung der Verbindung (z.B. bei Mouse-Over) wäre denkbar.

Testweise wurde die Darstellung von Relationen umgesetzt: Durch Setzen des Wertes `DRAW_RELATIONS = true` in den AppProperties kann dies aktiviert werden.

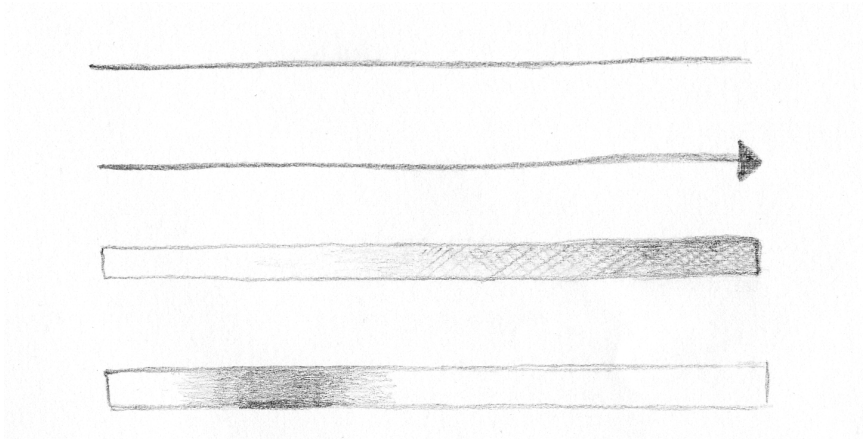


Abbildung 24 - Verschiedene Möglichkeiten der Darstellung von Relationen. Linie (ungerichtet), Pfeil (gerichtet), Band (gerichtet), Band (animiert)

Anzeigebedingungen

Da nicht alle Relationen zur gleichen Zeit angezeigt werden sollen, muss eine geeignete Situation bestimmt werden.

- Koppelung an LOD
- Aktivierung auf Verlangen des Nutzers per Mouse-Over oder Mouse-Click
- Ablesen des vom Auge fokussierten Objekts mittels zusätzlicher Hardware
- Wählen des Objektes als fokussiert, das im geringsten Abstand zum Mittelpunkt der Bildfläche liegt

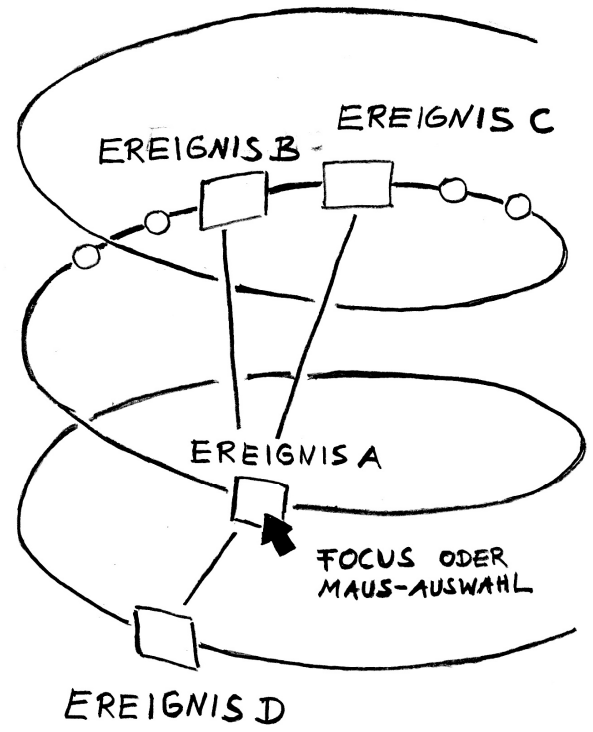


Abbildung 25 - Anzeige von Relationen auf Verlangen des Nutzers. Das sich im Focus befindliche Element zeichnet Verbindungslinien zu den mit ihm in Beziehung stehenden Elementen. Dabei gehen die verbundenen Elemente evtl. in eine höhere Detailstufe über, um diese hervorzuheben.

5.13 Head Up Display [+]

Ein HUD entzieht sich den perspektivischen Transformationen, liegt als Schicht über der Szene und sollte, um sich klar vom 3D-Raum abzuheben, flächig gestaltet werden.
Das Head Up Display kann für die Menüoptionen, Einstellung der Filter, die Auswahl der Darstellungsoptionen, Skalierungen und die Darstellung von inhaltlichen Zusatzinformationen genutzt werden. Außerdem lassen sich hier weitere Sichten auf die gerade betrachteten Inhalte anzeigen (siehe Abbildung 26 - HUD, unten links: Hier wird eine Überblicksperspektive eingeblendet, um die eigene Position im Gesamtzusammenhang zu kennzeichnen).

Abbildung 26 - HUD

5.14 Navigation

Die Navigation durch die Modelle ist per Tastatur und Maus möglich. Die Steuerung per Tastatur erlaubt eine präzise Führung entlang der Strukturen. Sie ist diskreter Natur, das einmalige Drücken einer Taste kann die Bewegung um eine bestimmte Einheit bedeuten, z.B. die Bewegung in der Zeit um genau einen Monat. Die Steuerung per Maus hat den Vorteil, stufenlos regulierbar zu sein, und kann genutzt werden, um den angezeigten Ausschnitt nachzujustieren. Nachteil der Mausnavigation ist die fehlende Genauigkeit. Zwei verschiedene Maus-Verhalten-Sets können eingestellt werden: PRIMITIVE und ADVANCED. Während PRIMITIVE die freie Bewegung im 3D-Raum ermöglicht, schränkt ADVANCED die Bewegung dahingehend ein, dass nur „Kopfdrehungen“ und Auf-Ab-Bewegungen möglich sind. Ungewollte Perspektiven werden so vermieden. Die Steuerung wird hier am Beispiel des Zeit-Helix-Modells beschrieben:

Tastatursteuerung für das Zeit-Helix-Modell

- `rechts`: geführte Bewegung entlang der Struktur mit der Zeit fortschreitend
- `links`: Bewegung zurück in der Zeit
- `vor`: Heranzoomen, Details nehmen zu
- `zurück`: Wegzoomen, Details nehmen ab

- `Alt` + `vor/zurück`: Bewegung vertikal in der Helix
- `Alt` + `rechts/links`: Drehung der Helix

Eingeschränkte Maussteuerung für die Helix

- `drag rechts/links`: Drehung der Helix
- `drag vor/zurück`: Bewegung vertikal in der Helix
- `Alt` + `drag vor/zurück`: Zoomen

Primitive Maussteuerung

- Freie Bewegung in alle Richtungen

Standard Blickwinkel

- `Strg+O`: Zurücksetzen zum Ursprung
- `Strg+N`: Ansicht aus etwas größerer Entfernung
- `Strg+H`: Horizontale Ansicht der Helix aus größerer Entfernung
- `Strg+T`: Zeitstrahl Ansicht der Helix aus größerer Entfernung
- `Strg+U`: Blick nach oben
- `Strg+D`: Blick nach schräg oben

5.15 Performanz

Zur Verbesserung der Performanz wurde die Wieder- und Mehrfachverwendung von Objekten, so oft es ging, eingesetzt, um Zeit beim Erzeugen neuer Objekte zu sparen und Speicherplatz zu teilen. Für größere Mengen Text in kleiner Schriftgröße empfiehlt sich die Verwendung von Text2D statt Text3D. Erstgenannter wird von Java3D als Bild gerendert, um Speicher einzusparen. Um die Anzahl und Komplexität der zu einem Zeitpunkt angezeigten Objekte überschaubar zu halten, wird Clipping und LOD verwendet. Beides wird von Java3D bereits geliefert.

Nachlademechanismus - LoadUnloadByDistance / LoadUnloadDelayByDistance

Da die Szene dynamisch aufgebaut wird und in der Regel nur ein Teil betrachtet werden soll, wird ein Nachlademechanismus (behaviors.LoadUnloadByDistance) verwendet, der benötigte Elemente, wie die speicherintensiven Bilder, erst dann lädt, wenn diese aktiviert werden und sie wieder aus dem Speicher freigibt, wenn diese passiviert werden. Das Aktivieren geschieht beim Eintritt der ViewPlattform des Betrachters in den Aktivierungsradius des Objekts. Das Passivieren kann entweder ebenfalls von der Entfernung zum Betrachter abhängig gemacht werden, oder zusätzlich mit Zeitverzögerung geschehen (behaviors.LoadUnloadDelayByDistance). Dies basiert auf der Überlegung, dass es durch Hin- und Herschauen oftmals passieren könnte, dass ein Objekt, das vor sehr kurzer Zeit passiviert wurde, wieder aktiviert wird. Solange die Objekte passiviert sind, wird ein Platzhalter-Objekt dargestellt.

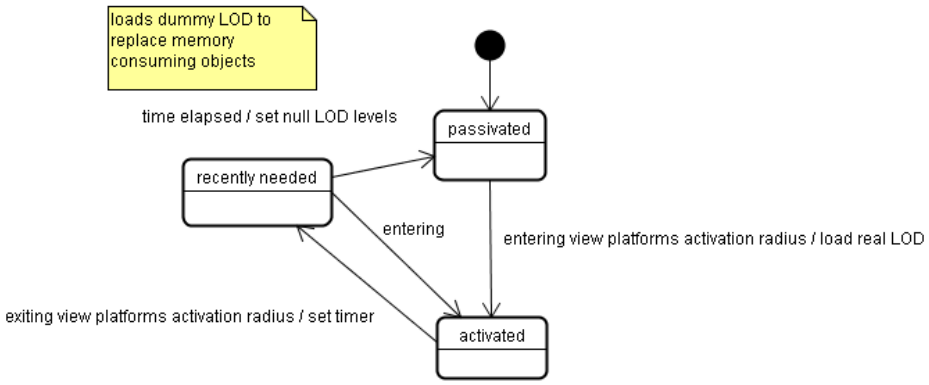


Abbildung 27 - State Chart Diagramm: Zustände beim LoadUnload-Behavior

Durch Setzen der LOD_BOUNDS lässt sich der Radius der Aktivierung ändern. Dieser sollte nicht zu klein gewählt werden, um ein unnötiges Freigeben zu verhindern.
Um ein langes Auslesen der Ontologie beim Starten der Anwendung zu vermeiden, wird diese nur auf Verlangen ausgelesen. Lediglich die Titel und die Datumsangaben, die zum Aufbau der Modellstruktur bekannt sein müssen, werden zu Anfang komplett ausgelesen. Alle anderen Eigenschaften der EntityRepresentation werden, wenn sie nicht in der EntityRepresentation gecachet sind, in diesem Moment aus der Ontologie ausgelesen (Proxy Pattern).

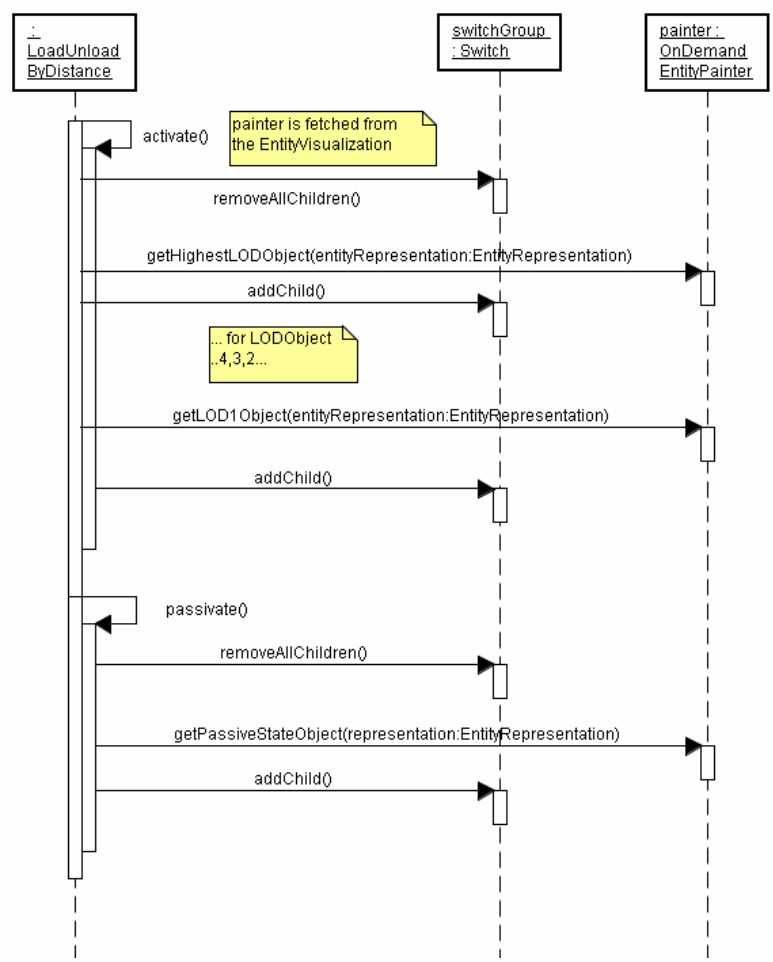


Abbildung 28 - Sequenzdiagramm: Aktivierung und Passivierung der EntityVisualizations

Für die einheitliche Behandlung von Behaviors, die die Performanz von LOD-Switches beeinflussen, wurde das Interface PerformanceBehaviorForSwitches eingeführt.

5.16 Gestaltung des Mediums

Halt bieten für die Orientierung im 3D-Raum

Um dem komplett dynamisch generierten Raum eine visuelle Festigkeit zu geben, lassen sich die unten folgenden Mittel einsetzen. Der Betrachter kann sich besser orientieren, wenn es eine Basis gibt, die sich nicht verändert und zu der man immer zurückkehren kann.

Struktur-Hilfselemente

Zweck der Struktur-Hilfselemente ist die Unterstützung des Betrachters beim Erkennen der Struktur. Die aus Elementen aufgebaute Struktur kann Lücken enthalten, die das Erkennen der Struktur erschweren, wenn diese zu groß werden. Struktur-Hilfselemente schließen diese optischen Lücken. Im Framework wird die Unterstützung der Struktur über den Aufruf der Methode drawVisualizationSupport() des Interface ModellStructure geregelt. So kann z.B. die Gestalt der Helix nachgezeichnet werden, damit für den Fall, dass (streckenweise) zu wenige Elemente vorhanden sind, die Struktur trotzdem sichtbar ist. Weiterhin kann auch das Anzeigen einer Skala denselben Effekt erzielen.

Fest stehende Objekte

Nicht alle Objekte in der Szene sollten beweglich sein. Einige optische Ankerpunkte, wie etwa eine Bodenfläche oder ein Hintergrund, können die Eigenbewegung des Betrachters gegenüber der Szene klarer machen. Andernfalls kann der Eindruck entstehen, dass sich die Szene bewegt und man selbst still steht. KULLBERG bezeichnet dies als das Problem der relativen Bewegung und verwendet bspw. Gitternetze um einen Boden erkenntlich zu machen. Über die Verwendung von Surroundings aus dem Package scene können solche Orientierungshilfen eingeblendet werden.

Kontinuierliche Übergänge [+]

Die anfängliche Überlegung, mit fließenden, animierten Übergängen bei sehr überladenen Modellen Ruhe zu schaffen, habe ich verworfen, denn der sprunghafte Wechsel von einem LOD zu einem anderem zieht oft weniger Aufmerksamkeit auf sich als ein animierter Übergang, der neue Bewegung ins Spiel bringt. Um einen unauffälligen Übergang zwischen den LOD zu ermöglichen, sollten sich die Level in möglichst wenigen Kriterien auf einmal ändern. Beispielsweise kann die Farbe und Größe, Position und ungefähre Form der dargestellten Objekte beibehalten werden.
Graduelle Änderungen kommen vor allem für sehr langsame Übergänge in Frage, wo die entstehende Bewegung, aufgrund ihrer Langsamkeit, nicht wahrgenommen wird, etwa das Einblenden weit entfernt liegender Partikel aus dem Hintergrund. Graduelle Änderungen können durch Überblendung, entweder über zunehmende/abnehmende Transparenz oder über die aktuelle Hintergrundfarbe erreicht werden. Die animierte Änderung von InfoBoards durch Aufklappen / Herausschieben wurde aus oben genannten Gründen ebenfalls verworfen.
Einsatzorte für kontinuierliche Übergänge sind neben den LOD-Wechseln das Einblenden und Ändern von Fokus und Relationen. Nach ZEHLER gehören solche kontinuierlichen Übergänge wie auch abrupte oder stufenweise Übergänge zu den visuellen Ereignissen.

Flächigkeit

Als Schriftarten empfehlen sich flächige Typen, wie „Arial Black“, die den Bildern und InfoBoards ähneln und so den optischen Wechsel von LOD zu LOD weiter erleichtern.

Metaphern

Zeitstrahl, Fortschritt

Die Metapher des Zeitstrahls und Fortschritts findet sich trotz der Zyklen, die die Helix aufweist, ebenfalls in ihr wieder. Die Y-Achse stellt einen Strahl dar, der auch per Navigation verfolgt werden kann (`ALT+oben/unten`).

Jahresringe

Wird eine Sicht in Y-Richtung verwendet (`Strg+U`), so ergibt sich eine Darstellung, die an Jahresringe erinnert, wobei die inneren Kreise jedoch je nach Blickrichtung für lang zurückliegende oder in der Zukunft liegende Zyklen stehen, so dass eine absolute Skala zur Orientierung hilfreich ist.

Atom [+]

Die Metapher des Atoms kann genutzt werden, die mit einem Entity in Verbindung stehende Objekte direkt in der Nähe anzuzeigen. Die Sicht auf den Originalkontext dieser Objekte wird dabei aufgegeben. Dies kann aber vor allem bei sehr weit auseinander liegenden Objekten oder, wenn der Kontext nicht als wichtig eingestuft wird, sinnvoll sein.
Wenn der Kontext der verbundenen Objekte dennoch sichtbar sein soll, kann von den „Elektronen“ des Atoms zusätzlich eine Verbindung zur eigentlichen Position im Raum gezeichnet werden.

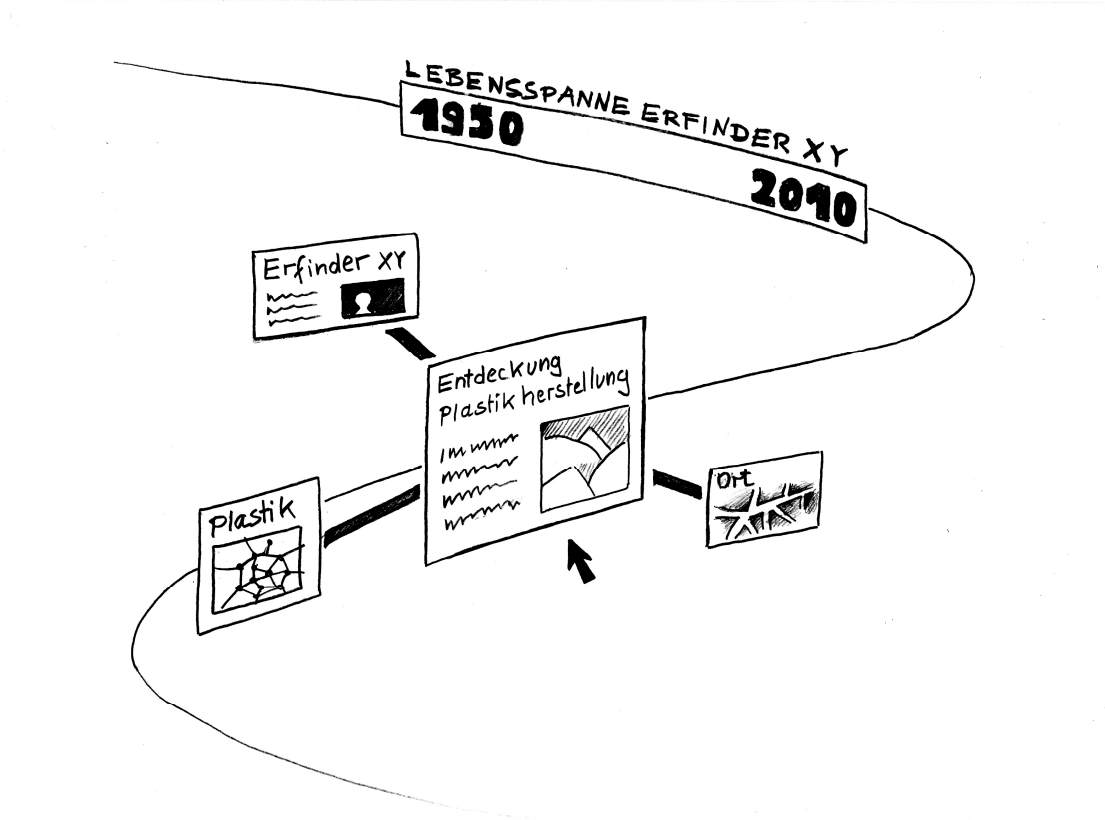


Abbildung 29 - Anordnung in Verbindung stehender Elemente nach der Atommetapher

Künstliche Perspektiven

Bei der Betrachtung des Helix-Modells ergeben sich sowohl 3D-Raum-Eindrücke als auch, begünstigt durch Billboard-Verhalten, flächige Ansichten. Oft entstehen diese dann, wenn eine sehr künstliche Perspektive, etwa exakt aus Y-Richtung oder exakt von der Seite, gewählt wird.
Die Einhaltung der Gesetze der Perspektive ist nur soweit nötig, wie es dem Betrachter hilft, die Orientierung im 3D-Raum zu behalten. Bis zu einem gewissen Grad können die Möglichkeiten der Virtualität gegenüber der Realität genutzt werden, wenn dies zur Erfassung der Informationen nützlich ist.
Dazu zählt das bereits erwähnte Billboard-Verhalten, aber auch Objekte fixer Größe, also Objekte, die nicht skalieren.
Durch Ankippen der InfoBoards zum Betrachter hin könnte z.B. das Problem gelöst werden, dass bei Blick in Y-Richtung die meisten Bilder nicht mehr sichtbar sind.
Der Wert der Anwendung ergibt sich durch die sorgfältige Abstimmung solcher Parameter.

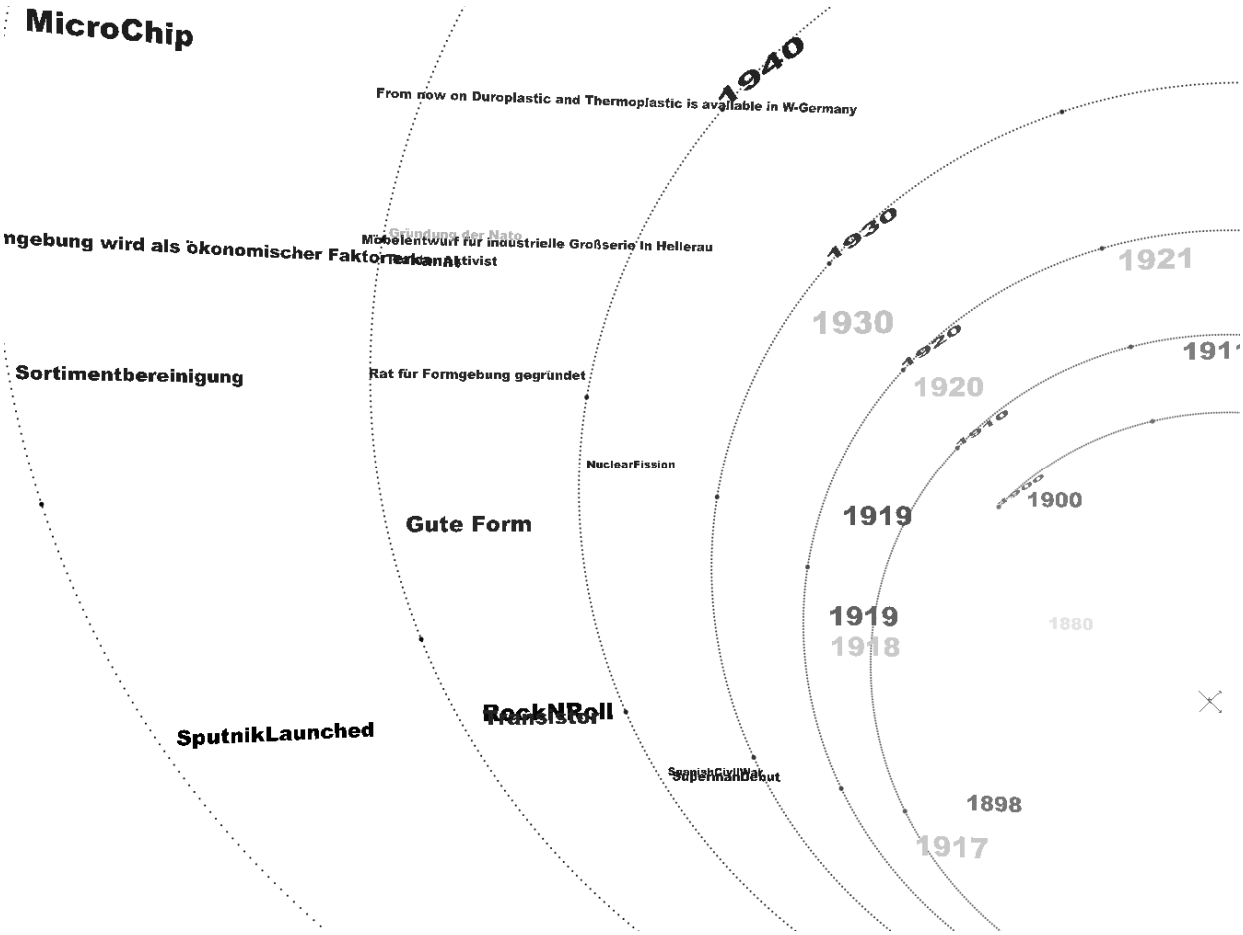


Abbildung 30 - flächig wirkende Perspektive

Farbperspektive und Tiefenschärfe

Die im Gestaltungsentwurf vorgesehenen Mittel Farbperspektive und Tiefenschärfe wurden versucht mit Java3D umzusetzen. Eine Farbperspektive lässt sich relativ leicht erzielen, indem man einen Nebel mit entsprechender, zum Hintergrund passender Farbe wählt und so einstellt, dass die weiter weg liegenden Elemente des Modells vom Nebel erfasst werden. Der Nebelbereich muss dabei erstens ausreichend groß sein, damit Objekte in den Nebelbereich fallen und zweitens vor dem Distanz-Clipping enden, damit alle Objekte vollständig ausgeblendet werden können.
Sowohl bei einer Umgebungsfarbe im Ton des Himmels als auch in der Farbe schwarz werden durch den Nebel weiter weg liegende Elemente langsam in der Sättigung reduziert und damit als weiter entfernt wahrgenommen.

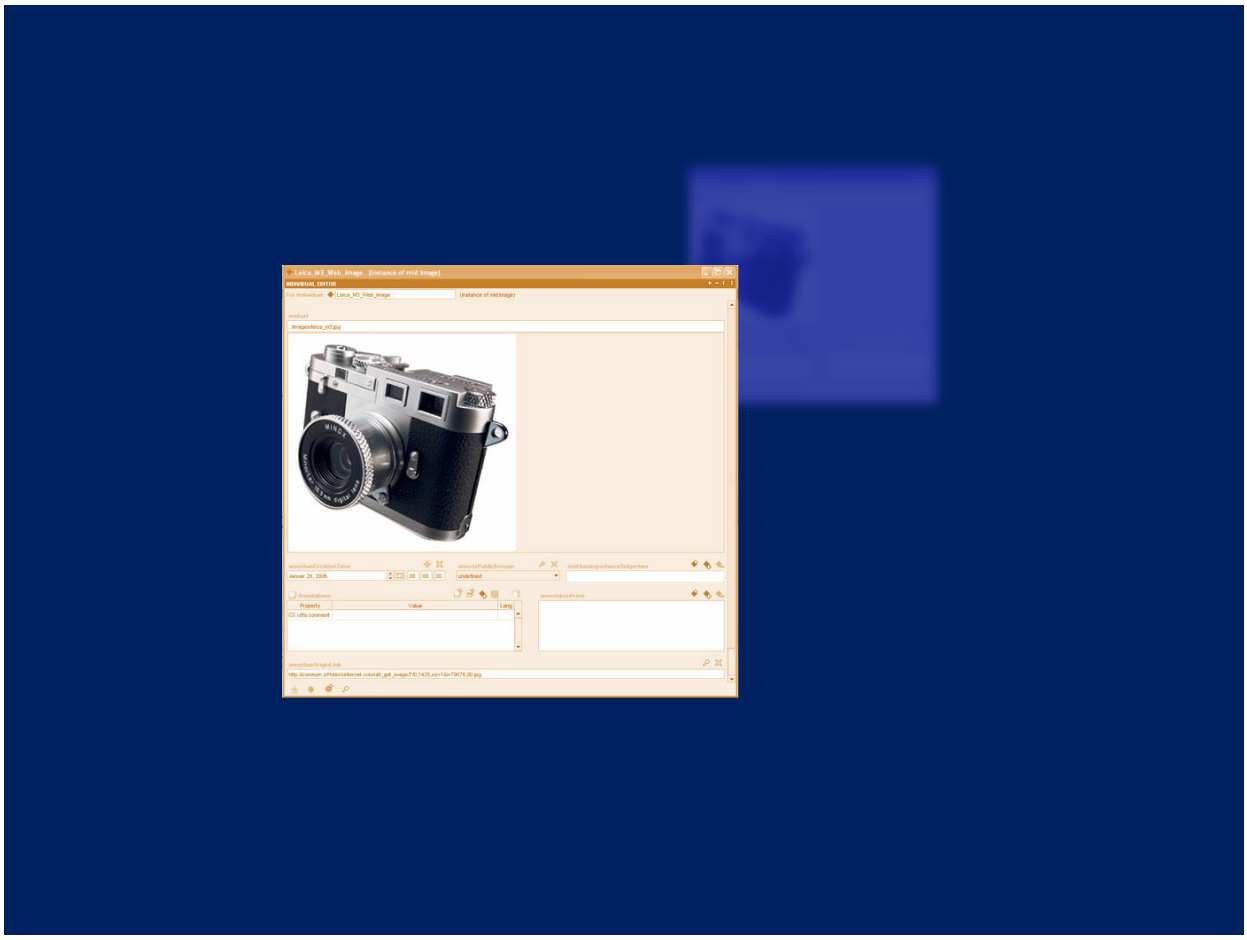


Abbildung 31 - Tiefenunschärfe und Farbperspektive

Tiefen(un)schärfe hingegen ist mit Java3D nicht ohne weiteres realisierbar, obwohl es von OpenGL [OPENGL] unterstützt wird. Vorraussetzung für einen solchen Mechanismus wäre in jedem Fall die Registrierung der Objekte, denen die Aufmerksamkeit des Nutzers gerade gilt. Sind diese bekannt, ließe sich die Ebene scharf darstellen, in der sie sich befinden. Bei der stereoskopischen Betrachtung erübrigt sich jedoch der Einsatz von Tiefenschärfe, da man sich nun tatsächlich auf einzelne Bildtiefen konzentrieren kann. Die dritte Dimension ist nicht mehr vermischt mit den anderen beiden Raumdimensionen.

Billboard

Als Billboard-Verhalten bezeichnet man in der Computergraphik die Eigenschaft von Objekten, sich dem Betrachter zuzuwenden. Ihre Orientierung im Raum ist also abhängig von der Position des Betrachters.

Je nach Einstellung lässt sich eine Drehung um einen Punkt und damit eine vollständige Orientierung zum Betrachter oder eine Drehung um eine Achse erzielen.
Billboard-Verhalten wurde massiv eingesetzt, um die Lesbarkeit von Texten aus verschiedenen Perspektiven zu garantieren. Hierbei muss ein Abwägen zwischen den konkurrierenden Zielen „Halt geben“ und „Gute Lesbarkeit“ stattfinden. Die Drehung um eine Achse lässt die Orientierung der Gesamtstruktur weiterhin erkennen, wohingegen die vollständige Orientierung eher zum Flächigen tendiert.
Java3D unterstützt das Billboard-Verhalten durch die Klassen OrientedShape3D und das Behavior BillBoard. Letzteres funktioniert jedoch nicht für mehrere Betrachter unabhängig, weshalb in den Klassen OrientedText2D und OrientedText3D Erstgenanntes zum Einsatz kommt. Bei der Erzeugung eines neuen Objekts dieser Klassen, kann per Parameter im Konstruktor angegeben werden, ob die Rotation zum Benutzer vollständig (um einen Punkt herum), oder um eine Achse geschieht. Die eingeschränkte Rotation bis zu einem bestimmten Winkel ist bisher nicht umgesetzt.

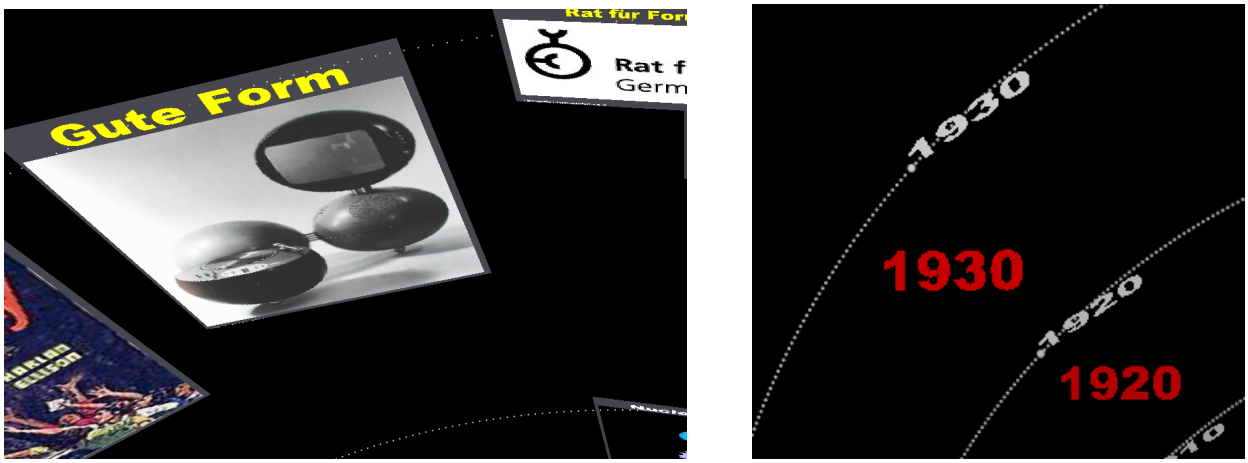


Abbildung 32 – Links kein Billboard-Verhalten. Rechts Rotation um Punkt (rote Schrift) und Achse (weiße Schrift)

6 Ausblick

Folgende Verbesserungen des Frameworks bestehen nur in der Idee und sind noch nicht umgesetzt oder näher untersucht worden:

Wissensbasis

- SPARQL Abfragesprache für Filtermechanismus nutzen statt direkt Jena anzusprechen

Verallgemeinerung

- Darstellung beliebiger Ontologien, die auf bestimmten Upper-Level-Ontologien aufbauen
- Diverse ModellStructures neben dem Helix-Modell
- Mapping graphischer Eigenschaften auf darzustellende Inhalte vom Nutzer einstellbar und automatische Berechnung der möglichen Einstellungen
- Unabhängigkeit von Java3D über Abstract Factory Pattern

Gestaltung

- Kontinuierliche LOD-Wechsel, wo sie sinnvoll sind

Performance

- Intelligenteres Lade-Verhalten (nicht alle LOD sofort laden, sondern erst wenn dieses LOD verlangt wird, bzw. ein oder zwei Level früher)
- Wiederverwertung der graphischen Elemente zwischen den LOD

Navigation

- Einheitliche Navigation für alle ModellStructures nach einem polymorphen Prinzip. Trotz unterschiedlicher Geometrie der Strukturen sollten wesentliche Merkmale, wie z.B. Hauptausrichtung, Ursprung und Größe per Interface von ModellStructure abfragbar sein, so dass die Navigation für verschiedene Modelle sich prinzipiell gleich verhält.
- Für die CAVE optimierte Navigation

7 Fazit

Der Gesamtumfang der Arbeit war im Voraus schwer abzuschätzen, da die Aufgabe auch darin bestand, verschiedene Wege auszuprobieren. Viele Überlegungen wurden gemacht, die sich letztendlich nicht im Resultat widerspiegeln. An vielen Stellen mussten Kompromisse eingegangen werden, um die gesamte Arbeitszeit, die für den Großen Beleg angesetzt war, nicht zu überschreiten. Die Eignung Java3Ds für die Graphik, sowie der Ontologien als Wissensspeicher, mussten sich erst zeigen.
Beim Durchführen der Belegarbeit hat sich der Wechsel zu einem prototypischen Vorgehen bewährt. Nach einer anfänglich sehr theoretischen Herangehensweise hat es sich als vorteilhaft erwiesen, direkt am Beispiel des Themas Designgeschichte zu arbeiten. Durch die Auseinandersetzung mit dem Thema Designgeschichte konkretisierten sich die Probleme und dadurch konnten neue Ideen aufgenommen werden. Unwesentliches wurde verworfen.
Auf diese Weise ist ein funktionierender Prototyp entstanden, der für die Präsentation der designgeschichtlichen Inhalte verwendet werden konnte. An diesem Prototyp haben sich viele Ideen des Gestaltungsentwurfs und neue Gestaltungsansätze als praktikabel bestätigt.
Java3D hat sich dabei als ausreichend in Bezug auf Funktionsumfang und Performance erwiesen. Auch das Anbinden einer Design-Ontologie als Backend für die Visualisierung funktioniert im Prototypen prinzipiell, auch wenn die entscheidenden Vorteile, die die Ontologie bietet, insbesondere die Filtermöglichkeiten, noch nicht ausgereizt werden.
Das Framework kann als solide Basis für weitere Experimente mit graphischen Parametern und Metaphern der Mediengestaltung dienen. Mit der Einführung von properties-Dateien zur zentralen Konfiguration gibt sich die Möglichkeit, verschiedene Varianten einer Visualisierung zu vergleichen. Dies ist vor allem vor dem Hintergrund nützlich, dass bereits kleine Details in der Umsetzung und feine Konfigurationsänderungen den Wert der Anwendung zur Visualisierung eines Themengebiets stark beeinflussen. Vielfach beeinflussen sich die Parameter gegenseitig, wodurch die korrekte Einstellung schwierig wird, so dass die Einführung von Abhängigkeiten zwischen den Parametern sinnvoll wäre.
Um mehr Allgemeingültigkeit zu erlangen und diverse Anwendungsgebiete zu unterstützen, muss vor allem ein dynamisches Mappingverfahren eingeführt werden. Außerdem sollte die Wahl der Upper-Level-Ontologie überprüft werden.

8 Anhang A – Installation

8.1 Vorraussetzungen

Voraussetzungen zur Inbetriebnahme der Anwendung sind:

- Installierte JRE ab Version 1.5.0_06
- Installiertes Java3D ab Version 1.3.1
- Mindest 512 MB RAM der Anwendung zugewiesen

8.2 Programmaufruf

Der Aufruf des Programms kann über den Aufruf
`javaw -Xms256m -Xmx512m universe.MainApp properties=standard.properties`
erfolgen. Über den Parameter properties wird eine Konfigurationsdatei angegeben, welche Angaben zu Anwendungsparametern enthält (siehe Seite 51, Konfiguration).

8.3 Stereoskopie

Java3D unterstützt die Generierung von einem Stereobild, wenn die Hardware dies zulässt. Folgende Stellen im Programmcode sind dafür relevant:

Einstellen des PhysicalBody: Java3dApplet, Methode createView()

```
…
PhysicalBody pb = createPhysicalBody();

// stereo settings
double halfEyeDistance = 0.0015;
pb.setLeftEyePosition(new Point3d(-halfEyeDistance,0.0d,0.0d));
pb.setRightEyePosition(new Point3d(halfEyeDistance,0.0d,0.0d));
…
```

Der Wert halfEyeDistance stellt den halben Augenabstand des Betrachters dar und ist für eine korrekte Differenz der beiden generierten Stereobilder verantwortlich. Je nach Aufbau der Stereoprojektion muss dieser Wert angepasst werden. Der Standardwert von 0.333 hat sich im Test-Aufbau als viel zu groß erwiesen. 0.0015 lieferte dagegen gute Ergebnisse.

Aktivieren der Stereobildgenerierung: Java3dApplet, Methode createCanvas3D( ):

```
…
Canvas3D c3d = new Canvas3D(SimpleUniverse.getPreferredConfiguration());
c3d.setSize( getCanvas3dWidth( c3d ), getCanvas3dHeight( c3d ) );

c3d.setStereoEnable(true);
…
```

Dies allein reicht jedoch noch nicht aus, denn stereo kann nur dann aktiviert werden, wenn die Bedingungen dafür gegeben sind. Per Parameter wird gefordert, dass stereo verwendet wird, wenn es möglich ist.

Ein Aufruf des Programms kann folgendermaßen aussehen:

```
javaw -Xms256m -Xmx512m -Dj3d.stereo=PREFERRED universe.MainApp
bzw. um stereo zu erzwingen:
javaw -Xms256m -Xmx512m -Dj3d.stereo=REQUIRED universe.MainApp
```

9 Anhang B – Beispielimplementierung zur Visualisierung des Themas „Geschichte des Designs in Deutschland im 19. und 20. Jh.“

Im Rahmen der Abschlussarbeit im Nebenfach Kunst und Gestaltung ist das entwickelte Framework dazu genutzt worden, die Designgeschichte Deutschlands im 19. und 20. Jahrhunderts darzustellen. Dabei wurde das Thema fortwährend bei theoretischen Überlegungen herangezogen, um diese am konkreten Beispiel zu überprüfen. Vielfach hat auch die Problematik des Themas zu neuen Frameworkentwicklungen geführt. Die Kombination der beiden Arbeiten auf diese Weise hat sich als sinnvoll erwiesen.

9.1 Eingrenzung des Umfangs

Anfänglich war es vorgesehen einen kompletten Abriss der bisherigen deutschen Designgeschichte zu machen. Angefangen im 18. Jahrhundert und weiter bis zu den aktuellen Entwicklungen. Dies hat sich als bei weitem zu umfangreich herausgestellt, so dass der Umfang auf das Kernthema Gegenüberstellung von Ost- und West-Design reduziert wurde, also vorrangig die Jahre 1945-1990 betrifft. Zusätzlich wurden einige herausragende Ereignisse aus den Jahren 1890-1945 hinzugenommen.

9.2 Überblick zur deutschen Designgeschichte

In den Jahren nach dem Zweiten Weltkrieg wurde überall in Deutschland improvisiert und die Formgestaltung hinten angestellt. Blech und Metallteile alter Waffen wurden von der Bevölkerung zu Küchenutensilien umgearbeitet.
Im Osten versuchte der Staat immer wieder die Entwicklung der Formgestaltung zu steuern. Anfangs um dem Kitsch zu begegnen und eine funktionale Gestaltung zu fördern, später dann um die mit der Gestaltung verbundene Kultur zu retten, die man durch den rein funktionalen Charakter bedroht sah. Die Kritik an dieser zwanghaften Orientierung des Designs fand Ausdruck in der Bezeichnung „Sozialistische Kaffeetasse“.
Später beeinflussten im Osten wie auch im Westen gesellschaftliche Themen die Diskussion um das Design wie z.B. die Gestaltung von Produkten für Behinderte und neue Technologien, die neue gestalterische Möglichkeiten mit sich brachten. Hier sind z.B. die Erfindung des Plastiks und des Cromargans zu nennen. In der DDR kamen im Zusammenhang mit den neuen Materialien die Diskussion um Gebrauchspatina, die Wegwerfmoral und die Kritik der Imitation auf.
Immer neue, kompliziertere Maschinen wurden entwickelt, deren Funktionsweise nur teilweise dem Benutzer sichtbar sein sollte, um ihn nicht zu verwirren. Die Idee der geschlossenen Form entstand und eine Schnittstelle zwischen Mensch und Maschine bildete sich heraus, die teilweise nichts von ihrer Funktionsweise mehr offenbarte, sondern als kastenförmige „Black box“ gebaut wurde. Diese wurde allerdings nach wenigen Jahren wieder durch das Prinzip der „Offenen Bauweise“ ersetzt, welche wieder mehr von der prinzipiellen Arbeitsweise der Dinge erahnen ließ, um den Gebrauch intuitiver zu machen. Durch die stetige Verkleinerung der funktionalen Bauteile und Verbreitung von Kunststoffen ließ sich die Form der Geräte vielfach beliebig gestalten. Hinter dem „Interface“ verborgen, wurde es für den Käufer schwieriger objektiv zu entscheiden. Im Westen wurde von der Bundesregierung die Stiftung Warentest gegründet, um dem Verbraucher eine Alternative zu einer Entscheidung auf rein optischer oder preislicher Basis zu bieten. Jedoch wurde nicht versucht, wie es im Osten der Fall war, unmittelbar auf die Designentwicklung Einfluss zu nehmen.
Im Osten wie im Westen entstanden Baukastensysteme, die der industriellen Fertigung entgegen kamen, aber trotzdem Flexibilität ermöglichen sollten. Diese Entwicklung umfasste die Erfindung der Schrank-

wand genauso, wie Werkzeug- und Bewässerungssysteme. Prädestiniert für die industrielle Fertigung, Lagerung und Verwendung als Komponente war die rechteckige Form. Eine weitere Entwicklung stellte die Gebrauchsgrafik dar. Systeme textloser Sinnbilder hatten ebenfalls einen modularen Aufbau und versuchten durch Einheitlichkeit eine schnelle Orientierung in Gebäuden oder bei der Benutzung von Maschinen zu ermöglichen.

Dies soll jedoch nur ein kurzer Überblick sein, denn das Ergebnis der Arbeit sollte kein neuer Text zur Entwicklung des Design sein, sondern der Versuch die vorhanden Fakten graphisch umzusetzen und so intuitiv Abfolgen von Ereignissen, deren Bedeutung und Relation zum weiteren Zeitgeschehen erfassen zu können.

9.3 Vorgehen

Um die in Textform vorliegenden Informationen zur Geschichte des Designs für die Visualisierung zu verwenden, musste zunächst untersucht werden, zu welchen Aspekten es Informationen gab (siehe auch Seite 18, Analyse der Domain Design). Solche Aspekte der Designgeschichte sind beteiligte Personen, Ausstellungen, das Aufkommen neuer Stilrichtungen und hervorstechende Produkte, die für bestimmte Entwicklungen repräsentativ waren.
Aus den Quellen, vorrangig „Designed in Germany (Since 1949)“ [ERLHOFF 1990] und „Gestalten für die Serie: Design in der DDR 1949-1985“ [HIRDINA 1988], wurden dann diese Informationen herausgesucht, im Zusammenhang gewichtet, und eine Auswahl in die Visualisierung übernommen.

9.4 Unscharfe Datumsangaben

Zu vielen Ereignissen wird nur die Jahreszahl in der Quelle angegeben, nicht jedoch das genaue Datum. Oft gibt es auch Angaben wie die „50er Jahre“ oder „Anfang des Jahrhunderts“, da das genaue Datum nicht von Interesse ist. Diese recht unscharfen Informationen stellen in einem Text keine Schwierigkeit dar – bei der Anordnung von diesen Ereignissen nach der Zeit kommt es jedoch dann sehr häufig zu Überlagerungen, weil die Ereignisse sich nicht über die beschriebene Zeitspanne verteilen, sondern alle am Anfang dieser Zeitspanne positioniert sind.
Bei der Angabe des Datums ist es momentan nötig, sich auf ein konkretes Datum festzulegen, da nur die Datumsangabe für die zeitliche Positionierung ausgewertet wird. Es ist zwar möglich, eine das Ereignis umschließende Zeitspanne wie etwa die „50er-Jahre“ anzugeben, allerdings wird darauf noch nicht vom Programm zugegriffen. Beim Fehlen eines konkreten Datums sollte das Programm entscheiden dürfen, wie die Verteilung gleichzeitiger Ereignisse vorzunehmen ist. Dabei sollten auf jeden Fall auch die kausalen Zusammenhänge mit berücksichtigt werden, die zwischen „gleichzeitig“ eingetragenen Ereignissen bestehen. Ist ein Ereignis Ursache eines weiteren, so sollte sich das in der Anordnung der Informationselemente widerspiegeln (siehe auch Seite 66, Häufung von Ereignissen).

9.5 Kontextereignisse

Um einen Kontext für die Designereignisse zu bieten, wurden für jedes Jahrzehnt ca. vier bedeutende Ereignisse gewählt. Die Bedeutung beruht auf meiner subjektiven Einschätzung, stellt jedoch eine Auswahl aus Ereignissen dar, die auf den Wikipedia-Seiten zu Jahrzehnten, in kollektiver und somit recht repräsentativer Weise, zusammengetragen wurden.
Bei der Auswahl der Ereignisse habe ich wert darauf gelegt, dass die Ereignisse einerseits einen gewissen Wiedererkennungswert, bezogen auf die Epoche, haben. Also mit dem Ereignis sofort eine gewisse Epoche verbunden wird und bereits vorhandenes Geschichtswissen aktiviert werden kann. Dazu war es zusätzlich wichtig, bei der Auswahl der Bilder für diese Ereignisse solche zu verwenden, die repräsentativ

und bekannt sind. Vorteilhaft ist außerdem, wenn sie sich bereits aus größerer Entfernung erkennen lassen.
Zum anderen sollten Ereignisse mit aufgenommen werden, die einen starken Bezug zum Design haben, also Designereignisse unmittelbar beeinflusst haben. Solche Ereignisse finden sich vor allem im Bereich neuer Technologien, Verfahrenstechniken sowie auch einschneidenden gesellschaftlichen Veränderungen. Politische Ereignisse beeinflussen das Design ebenfalls, allerdings erst indirekt über die gesellschaftlichen Veränderungen zu denen sie führen.

9.6 Ursache-Wirkung-Beziehungen

Zwischen den Ereignissen der Designgeschichte gibt es oft Ursache-Wirkung-Beziehungen welche von der Anwendung über Verbindungslinien dargestellt werden können. (siehe auch Seite 69, Darstellung von Relationen). Solche Beziehungen gibt es z.B. zwischen dem Ereignis „Erfindung der Waschmaschine“ und dem Ereignis „Waschtag wird überflüssig“. Sie ergeben sich aus dem Zusammenhang des Textes und sind ein wichtiger Bestandteil der Information, die bei der Visualisierung von geschichtlichen Daten berücksichtigt werden muss. Fehlen sie, gehen diese Zusammenhänge verloren, lediglich eine Aneinanderreihung von Ereignissen nach der Zeit bleibt übrig. Im Text sind solche Beziehungen oft an Formulierungen wie „durch…kam es zu…“, „… führte zu…“ oder „die Bedingung für … war…“ gebunden. Die in der Beispielimplementierung dargestellte Ursache-Wirkung-Beziehung[3] ist eine sehr generelle Beziehung. Spezialisierungen wie „A istVorbildProduktfür B“ oder „A war Vorraussetzung für B“ sind hier eingeschlossen[4].

9.7 Mehrsprachigkeit

Über das Setzen der Einstellungen

```
FIRST_LANGUAGE = <Sprach-Kürzel> und
SECOND_LANGUAGE = <Sprach-Kürzel>
```

in den Konfigurations-Dateien lässt sich die Sprache der ausgegebenen Titel, Beschreibungen, und sonstiger Texte steuern. Dabei versucht das System zunächst eine Entsprechung in der als FIRST_LANGUAGE angegebenen Sprache zu finden. Ist dies nicht erfolgreich, wird auf SECOND_LANGUAGE, sonst auf eine andere gefundene Sprache zurückgegriffen. Wird für den Titel eines Ereignisses gar keine Bezeichnung gefunden, so wird der interne Name des Ereignisses verwendet.

9.8 Quellenangaben

Angaben zur Herkunft von Bildern sind direkt am Bild notiert. Es gibt die Möglichkeit, zu jeder Quelle (Bilder, Texte) eine oder mehrere Quellenangaben in Form von Literaturhinweisen hinzuzufügen. Diese Literaturhinweise bestehen aus dem Verweis auf ein Buch, das gesondert abgespeichert wird und einer Seitenzahlangabe. Auf diese Weise können direkt beim Übernehmen neuer Quellen in die Wissensbasis Angaben zu Copyright und Herkunft gemacht werden. Durch die Aufteilung in Seite und Buch muss nicht jedes Mal aufs Neue das Buch eingegeben werden. Dies kommt einer praktikablen Benutzung zu Gute.

---

[3] in der Ontologie als hist:effects definiert
[4] in der Ontologie als Property-Vererbung realisierbar

[+] Weil die Quellenangaben direkt an der jeweiligen Ressource vermerkt sind, könnten diese leicht auch in der Visualisierung, z.B. als Bildunterschrift bei Mouse-Over, angezeigt werden. Dies ist bisher nicht umgesetzt, ließe sich aber ohne großen Aufwand realisieren.

9.9 Bildmaterial

Die verwendeten Bilder sind größtenteils aus den betreffenden Büchern gescannt und somit nicht frei von Urheberrechten. Teilweise wurden die Bilder beschnitten sowie die Helligkeit und der Kontrast aufgearbeitet um eine klarere Darstellung zu erzielen.
Die Bilder für die Kontextinformationen stammen, wenn nicht anders vermerkt, aus Wikipedia und sind somit in der Regel frei verwendbar.

Literaturverzeichnis

[BRADE POLOWINSKI 2004] Brade, Marius / Polowinski, Jan. Informationsdarstellung in 3D-Räumen. Komplexpraktikum Mediengestaltung. August 2004.

[ERLHOFF 1990] Erlhoff, Michael. Designed in Germany. Since 1949. Prestel-Verlag, München, 1990.

[EYE_SYS] http://www.eye-sys.com

[GAMMA_ET_AL 1996] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software. Addison-Wesley, 1996

[GOULD 1987] Gould, Stephen Jay. Die Entdeckung der Tiefenzeit. Carl Hanser Verlag München.

[GROH FRANKE 2005] Farbperspektive im Kontext von Navigation durch virtuelle Welten - Artikel zu den theoretischen Grundlagen der Interfacegestaltung Rainer Groh und Ingmar S. Franke Technische Universität Dresden, Fakultät Informatik Institut Software- und Multimediatechnik, Lehrstuhl Mediengestaltung Prof. Dr. habil R. Groh, D-01062 Dresden

[HIRDINA 1988] Hirdina, Heinz. Gestalten für die Serie : Design in der DDR 1949-1985. [Hrsg.]. Amt für industrielle Formgestaltung und Bauhaus Dessau. Dresden : VEB Verlag der Kunst, 1988. - 395 S. : Ill.

[HORWICH 1987] Horwich, Paul. Asymmetries in time. 1987. Massachusetts Institute of Technology.

[HYPERHISTORY] http://www.hyperhistory.com

[JAMBALAYA] Jambalaya. http://www.thechiselgroup.org/?q=jambalaya

[JAVA3D] Java3D. http://www.j3d.org/

[JENA] Jena. http://jena.sourceforge.net/

[JUDE] Jude™. http://jude.change-vision.com/jude-web/index.html

[KULLBERG 1996] Kullberg, Robin L. Dynamic timelines: visualizing the history of photography. ACM 1996. http://doi.acm.org/10.1145/257089.257388

[OPENGL] OpenGL. http://www.opengl.org/

[OWL] Web Ontology Language. http://www.w3.org/TR/owl-features/

[PRIETO 1991] Prieto-Díaz, Rubén. Software. 1991. http://doi.acm.org/10.1145/103167.103176

[PRISS 2000] Priss, Uta. Faceted Information Representation. 2000. http://citeseer.ist.psu.edu/priss00faceted.html

[PROTEGE] Protégé. http://protege.stanford.edu/

[RDF] Resource Description Framework. http://www.w3.org/RDF/

[RDFS] RDF Schema. http://www.w3.org/TR/rdf-schema

[SEMANTIC_WEB] Semantic Web. http://www.semanticweb.org/

[SUMO] http://ontology.teknowledge.com

[SUO_KIF] SUO-KIF. http://suo.ieee.org/SUO/KIF/suo-kif.html

[TOP_BRAID] TopBraid™ Composer. http://www.topbraidcomposer.com

[VISUAL_MINING] http://www.visualmining.com

[W3C] World Wide Web Consortium. http://www.w3.org/

[ZEHLER 2004] Zehler, Thomas. Software-Visualisierung. (BTU) Report: ViSEK/045/D Version 1.30 04.02.2004

Glossar

Abbildungsverzeichnis