Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik

Bachelor-Thesis

im Studiengang
Business Information Systems

Thema:
Ein grafischer Editor für Transformationen zur Produktion semantischer Relationen

Unternehmen:

Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme (IAIS)

eingereicht von: Peter Eiser ‹peter-eiser@smail.inf.h-brs.de›

Matrikel Nummer: 9009053

eingereicht am: 14. Oktober 2011

Erstprüfer: Herr Prof. Dr. Martin Eric Müller
Zweitprüfer: Herr Dr. Kai Stalmann

Als Hauptziel dieser Arbeit soll untersucht werden, inwieweit der komplexe Prozess des Metadaten-Mappings mit XSLT-Skripten durch einen grafischen Editor unterstützt werden kann. Zu diesem Zweck soll ein Editor-Prototyp entwickelt werden, mit dessen Hilfe Mappings von XSLT-Parametern auf RDF-Tripel grafisch bearbeitet werden können.

In einer ersten Untersuchung hat sich gezeigt, dass die Entities und Properties der Ontologie der Deutschen Digitalen Bibliothek (DDB) in einer für Benutzer verständlichen Form als Knoten und Kanten eines Graphen repräsentiert werden können. Auf der Basis dieses Ergebnisses wurde die technische Umsetzung des Editor-Prototyps konzipiert und implementiert. Eine Hauptforderung dabei war eine saubere Trennung von Datenmodell und grafischer Oberfläche.

Der im Rahmen dieser Arbeit entwickelte Editor-Prototyp ist in der Lage, die geforderten Mappings zu erzeugen. Zur Unterstützung des gesamten Mapping-Prozesses ist jedoch noch eine weitere Ausbaustufe erforderlich.

Danksagung

Zuerst danke ich Dr. Robert Mertens für die Ermöglichung dieses Projekts. Er hat mich als Betreuer sowie Zweitprüfer während der Anfertigung der Arbeit begleitet und stand mir mit Tipps und Anregungen immer zur Seite gestanden. Da er die Chance für ein Auslandsjahr in Amerika am International Computer Science Institute Berkeley wahrnahm, hat Dr. Kai Stalmann diese Aufgabe übernommen. Ich danke Dr. Stalmann und Dr. Mertens für dieses interessante Thema. Dr. Stalmann gab mir wertvolle Hinweise und hatte immer ein offenes Ohr für Fragen im technischen Bereich. Mein Dank gilt auch Prof. Dr. Martin Eric Müller, der an der Hochschule die Betreuung für mich hatte und mir bei Problemen immer eine Hilfe war. Wenn ich Fragen zum DDB-Projekt hatte, konnte ich mich immer an Thorsten Wunderlich und Christoph Tornau wenden, sowie an Hermann Josef Hill. Vielen Dank für die tatkräftige Unterstützung und ihre Hilfe als Lektoren. Für die Korrekturarbeiten während der Erstellungsphase möchte ich Susanne Eiser, Sabrina Eiser, und Sandra Mechlinski danken. Meiner Frau Susanne und meinen Kindern danke ich, dass sie viel Geduld und Interesse hatten und mich während des ganzen Zeitraums unterstützten.

Eidesstattliche Erklärung

Ich versichere an Eides statt, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum *Unterschrift*

Inhaltsverzeichnis

Abkürzungsverzeichnis

ASC	Augmented SIP Creator
CIDOC CRM	CIDOC Conceptual Reference Model
DC	Dublin Core
DDB	Deutschen Digitalen Bibliothek
DOM	Document Object Model
DOM	Documenten Object Model
FIZ	Leibniz-Institut für Informatik
KWE	Kultur- und wissenschaftlichen Einrichtungen
OAIS	Offenes Archiv-Informations-System
RDF	Resource Description Framework
SAX	Simple API for XML
SGML	Standard Generalized Markup Language(Normierte Verallgemeinerte Auszeichnungssprache)
SIP	Submission Information Package
SWT	Standard Widget Toolkit
XML	Extensible Markup Language
XPath	XML Path Language
XSL	Extensible Stylesheet Language

Listings

Abbildungsverzeichnis

1. Einleitung

1.1. Fraunhofer-Gesellschaft

Die Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e. V., ist das größte Forschungs-Institut in Europa, mit mehr als 18000 Mitarbeitern an 80 Forschungseinrichtungen. Ihr Forschungsvolumen beläuft sich auf 1,65 Milliarden Euro zu Teilen aus Leistungsbereichen sowie einer Grundfinanzierung von Bund und Ländern. Die Hauptniederlassung befindet sich in München [vgl. FG11].

Das Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme (IAIS) mit Sitz in Sankt Augustin am Schloss Birlinghoven [vgl. Fra11] beschäftigt ca. 260 Mitarbeiter. Es ist für die Entwicklung des Projekts „Deutsche Digitale Bibliothek “ (DDB)[1] beauftragt worden (siehe Kapitel 2.2). In Folge der Umsetzung des DDB-Projektes entstand die Problemstellung, die zur dieser Arbeit geführt hat.

1.2. Problemstellung

Das DDB-Projekt hat als Ziel, auf Basis der Metadatenbestände der 30 000 Kultur- und wissenschaftlichen Einrichtungen (KWE) in Deutschland (Bibliotheken, Sammlungen und Archive) ein Wissensnetz zu erstellen. Diese Bestände liegen in den unterschiedlichsten Formaten vor. Die Verwaltung eines solchen heterogenen Datenbestandes in einem IT-System ist jedoch äußerst problematisch, da z. B. die Suche für alle vorhandenen Datenformate verschieden implementiert sein müsste. Daher ist eine Überführung (Mapping) in ein einheitliches Datenmodell sinnvoll. Die Regeln dafür werden aktuell von Hand programmiert und müssen bei jeder Änderung von einem Spezialisten angepasst werden. Dies geschieht mit der Extensible Markup Language (XML) und der Transformations-Sprache XSLT (siehe Kapitel3.2). Die Entwicklung bzw. Anpassung eines Mappings ist ein komplexer Vorgang (siehe Kapitel 4.2), der von den einzelnen KWE nur schwer zu leisten ist. Aus diesem Grund soll untersucht wer-

[1] Siehe Kapitel 2

den, ob dieser Vorgang mit einem grafischen Editor unterstützt werden kann. Hierzu soll ein solcher Editor, mit dem Transformations-Templates zur Produktion semantischer Relationen erstellt werden können, prototypisch konzipiert und implementiert werden.

1.3. Motivation

Die Verwendung der Transformations-Templates ist Teil der Input-Datenaufbereitung (Pre-Ingest) und im Augmented SIP Creator (ASC) verankert. Der ASC ist eine Komponente des DDB-Projekts und wird den Kultureinrichtungen zur Verfügung gestellt. Er transformiert die Daten der Kultureinrichtungen in das DDB-eigene Format und bereitet so den Datenimport (Ingest) vor. Die Produktion von XSLTs ist ein nicht trivialer Prozess, der ein interdisziplinäres Wissen voraussetzt. Der Kenntnisstand umfasst folgende Techniken: Metadatenformaten, Ontologien wie das CIDOC CRM, Java, XML, XSLT, RDF. Aus diesem Grund entstand diese Abschlussarbeit, um zu überprüfen, ob die Erstellung der XSLTs mit einem grafischen Editor möglich ist. In diesem Editor muss ein Graph zu konstruieren sein, der der DDB eigenen Ontologie entspricht und als Resultat die benötigten XSLTs hat. Die Komplexität soll sich so auf ein Minimum reduzieren.

1.4. Ablauf der Arbeit

Für die Konzeption des Editor-Prototyps sind im ersten Schritt mehrere grundlegende Entscheidungen zu treffen:

1. Konzeption der Methode zur technischen Umsetzung der Ontologie
2. Konzeption der technischen Umsetzung der Ausgabe des Transformations-Skripts
3. Konzeption der technischen Umsetzung der grafischen Oberfläche
    - Auswahl der Programmierschnittstelle (API) zur grafischen Benutzerschnittstelle (GUI).
    - Auswahl des grafischen Frameworks zur Darstellung des Graphen

Im nächsten Schritt wird die Logik implementiert und darauf die GUI entwickelt. Bei der Implementierung der GUI soll möglichst das MVC[2] Muster zur Anwendung kommen, da eine Weiterentwicklung seitens anderer Stelle so barrierefrei wie möglich sein soll. Im nächsten Schritt erfolgt die detaillierte Beschreibung und die Evaluierung des Editors.

[2]MVC Muster besteht aus Model (Daten), View (Ansicht), Controller (Steuerung) [vgl. BP08, S. 161]

1.5. Überblick

Im Kapitel 2 wird das DDB-Projekt beschrieben, zum einem aus der organisatorischen, aus der technischen Sicht und zum anderen die Architektur, insbesondere wird hier auf den Datenimport eingegangen. Das Kapitel 3 gibt einen Überblick über die Techniken, die zur Umsetzung der Arbeit benötigt werden. Des weiteren beschäftigt sich das Kapitel 4 mit dem Augmented SIP Creator (ASC) und mit dem Kapitel 5 wird die Konzeption des XSLT RDF Editors erläutert.

2. Das Projekt „Deutsche Digitale Bibliothek“

Mit diesen Kapitel wird ein kurzer Überblick der für diese Arbeit relevanten Teile des DDB-Projektes gegeben.

2.1. Ziel der DDB

Bund, Länder und Kommunen haben mit der Deutschen Digitalen Bibliothek (DDB) ein IT-Projekt geschaffen, das der Allgemeinheit Kulturgüter und wissenschaftliche Informationen über das Internet zugänglich machen soll. Mit Ausnahme der Werke, die in irgendeiner Weise (Urheber-) rechtlich geschützt sind, soll das Portal zur kostenfreien Nutzung zur Verfügung gestellt werden [vgl. Min09b, S. 8 f.]. Indem die DDB in Deutschland ca. 300 Millionen Objekte von 30.000 Kultureinrichtungen zusammenfasst und zu einem Wissensnetz verknüpft, fördert sie das Wissen und die Information in der Gesellschaft [vgl. Min09b, S. 8 f.].

2.2. Rahmenbedingungen der DDB

Das DDB-Kompetenznetzwerk wurde als gemeinsames Projekt von Bund und Ländern ins Leben gerufen. Beide fungieren auch als Träger des Projekts [vgl. Min09b, S. 19]. Das Kompetenznetzwerk übernimmt die Aufgaben des Aufbaus und den Betrieb der DDB [vgl. Min09a, S. 15]. Die Finanzierung wird von Bund und Ländern je zur Hälfte getragen, die Länder tragen die Finanzierung gemäß des Königsteiner Schlüssels[1]. Für die Entwicklung wurde das Fraunhofer-Institut IAIS beauftragt. Den Betrieb übernimmt das Leibniz-Institut für Informatik FIZ [vgl. DNB11].

[1] Der Königssteiner Schlüssel ist in Art.91b Abs.3 GG verankert und ist ein jährlich neu berechneter Schlüssel zur Verteilung der Finanzierung wissenschaftlicher Forschungsprojekten unter Bundesländern[vgl. Min09a, S. 15]

2.3. Benutzer der DDB

Es gibt drei Benutzergruppen für die DDB. Das sind die Endnutzer, Kultur- und Wissenseinrichtungen und Unternehmen (Kulturwirtschaft). Endnutzer nutzen das System über eine Browser basierte Schnittstelle um auf dem Portal nach Informationen zu suchen. Kultur- und Wissenseinrichtungen stellen Inhalte zur Verfügung und brauchen hierfür eine Softwarekomponente, den Augmented SIP Creator (ASC), mit dessen Hilfe die heterogenen Eingangsformate in das DDB-interne Format (SIP) transformiert und in das System importiert werden. Dies soll nur auf die Kultur- und Wissenseinrichtungen aus Deutschland beschränkt sein. Die Nutzungsberechtigung wird vom DDB-Kompetenznetzwerk vergeben. Unternehmen, die Informationen aus dem Kulturbereich benötigen, kann die DDB als Lieferant dienen [vgl. DDB10, S. 44 f.].

2.4. Konzept der DDB

Die DDB-Plattform besteht aus zwei internen Komponenten und einer externen. Die internen Komponenten sind der Kern der digitalen Bibliothek und das Portal (Präsentation, Verwaltung und Personalisierung). Die externe Komponente ist der ASC, der als eigenständiges Programm die heterogenen Eingangsformate für den Ingest vorbereitet. In der Plattform werden die Daten ingestiert, indiziert und redundant gespeichert. Die Komponente namens Object Discovery ist für die Suche, das Browsen und die objektspezifischen Darstellungen zuständig [vgl. Su11, S. 8 f.].

2.5. Architektur und Implementierung der DDB

Die Architektur ist in einem Endstadium, als Quelle dient hier die Abhandlung von [vgl. Su11, S. 4 f.]. Die DDB ist nach dem Prinzip der SOA[2] aufgebaut. Sie teilt sich in drei Schichten auf, die Präsentationsschicht, die Serviceschicht und die Speicherschicht. Die Implementierung der DDB ist zu diesem Zeitpunkt noch nicht abgeschlossen, so dass sich diese Beschreibung auf den Stand zu Beginn dieser Arbeit bezieht.

[2] Service-Oriented Architecture (engl. für dienstorientierte Architektur) ist ein Architekturmuster für die Softwareentwicklung, bei dem jede Funktionalität, bzw. jeder Geschäftsprozess eines Unternehmens, durch einen oder mehrere komponierte Services (engl. für Dienste) ausgeführt werden kann [vgl. FH08, S. 834 f.].

2.6. Datenquellen der DDB

Die eingehenden Metadaten beschreiben digitale Objekte (Digitalisate und digital-born Objekte) und binären Content (Bilder, Audiodateien, Filme etc.). Die Metadaten sind in verschiedenen Formaten kodiert (Dublin Core (DC), LIDO und EAD etc.). Um sie in die DDB zu überführen, werden die Informationen aus den Metadaten als RDF-Tripel extrahiert und gespeichert [vgl. Su11, S. 4 f.].

2.7. Input-Datenaufbereitung

Dieser Abschnitt enthält die Beschreibung des Datenimports sowie der Datenaufbereitung für die Plattform. Der Datenimport aus den Kultureinrichtungen erfolgt in zwei Schritten (siehe Abb. 2.1). Im ersten Schritt, dem Pre-Ingest, werden die Metadaten der einzelnen Kultureinrichtungen in das DDB-Modell transformiert. Diese Aufgabe übernimmt der ASC, der jeder KWE zur Verfügung gestellt wird. Er muss verschiedene Metadatenformate in das DDB-interne Format übertragen und diese Transformation zu einem SIP muss so erfolgen, dass die Informationen über die Objekte des kulturellen Erbes erhalten bleiben. Das interne Datenmodell ist an das CIDOC CRM angelehnt (siehe Kapitel 3.4). Die so erstellten Eingangsdaten müssen einerseits so aufbereitet werden, dass sie von Benutzern verstanden werden können (HTML-Ansichten), andererseits müssen ihre Inhalte maschinenlesbar sein (als RDF-Beschreibung). Die Aufbereitung der heterogenen Eingangsformate zu einer RDF-Beschreibung erfolgt durch Metadaten-Ontologie-Mapping. Hierzu werden Extensible Stylesheet Language Templates (XSLT) zur Transformation eingesetzt [vgl. Su11]. Der ASC besitzt zum jetzigen Zeitpunkt nur standardisierte Transformationen für bestimmte Metadatenformate. Unterschiede bestehen jedoch nicht nur zwischen den Metadatenformaten, sondern auch in ihrer Interpretation und Verwendung durch die einzelnen KWE. Aufgrund begrenzter Ressourcen können Anpassungen für die einzelnen KWE nicht im Rahmen des DDB-Projekts erfolgen, sondern müssten durch die KWE in Eigenregie durchgeführt werden. Aufgrund der Komplexität dieser Aufgabe ist es angedacht, zu testen, ob die Erstellung der XSLTs mit einem Programm möglich ist [vgl. Su11].
Im zweiten Schritt, dem Ingest, werden dann im Kern der Bibliothek (Cortex) aus den SIPs Archival Information Packages (AIP) erzeugt (siehe Abb. 2.1). Die AIPs dienen dann zur Befüllung des Triple-Stores, des Archiv-Stores und des Index.

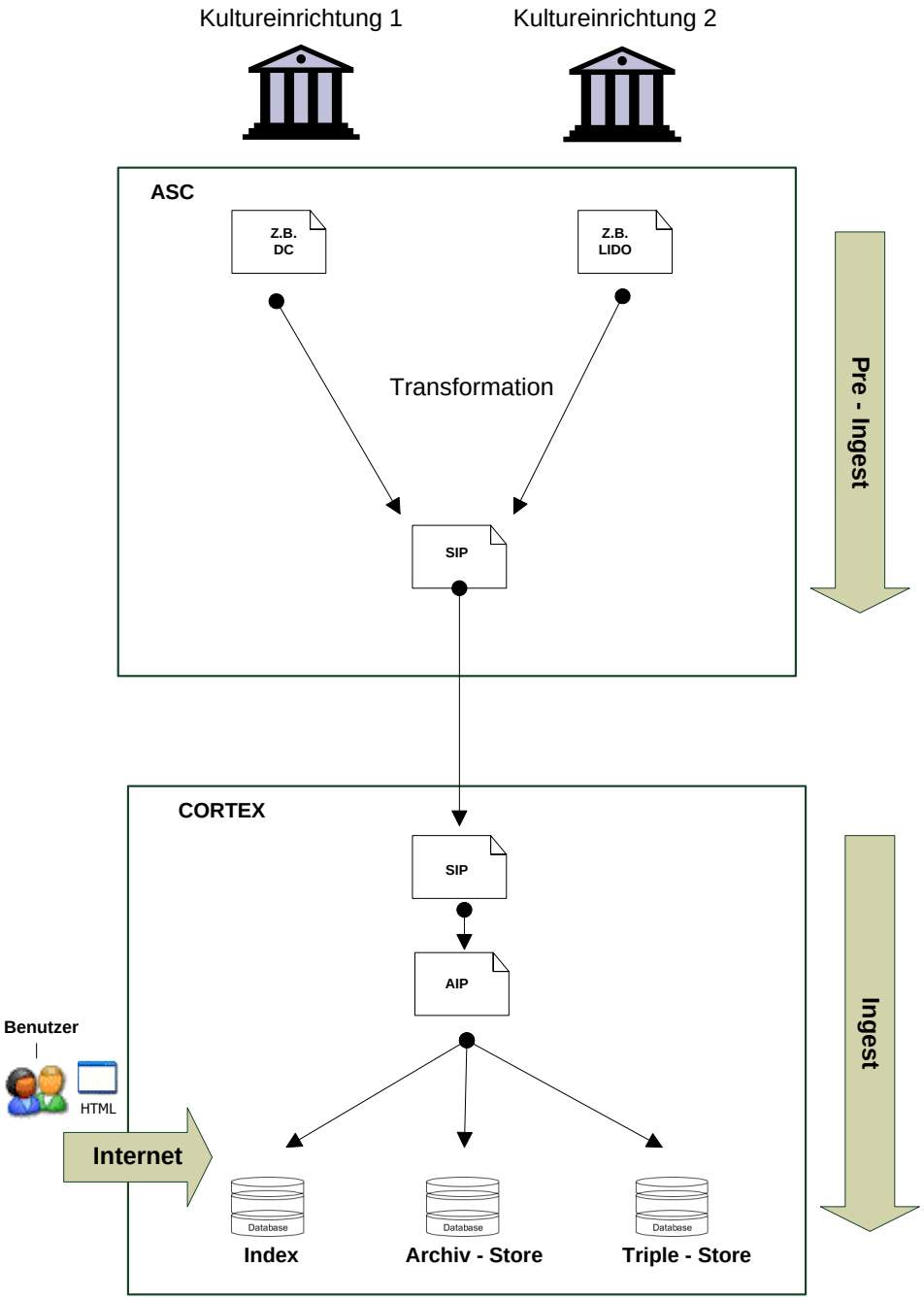

Abbildung 2.1.: Input-Datenaufbereitung

2.8. Datenimport

Der Datenimport basiert auf dem OAIS[3]-Modell und beginnt mit der Aufnahme des SIP. Darauf folgt die Umwandlung des SIP ins AIP. Abbildung 2.2 zeigt den Fluss der Metadaten.

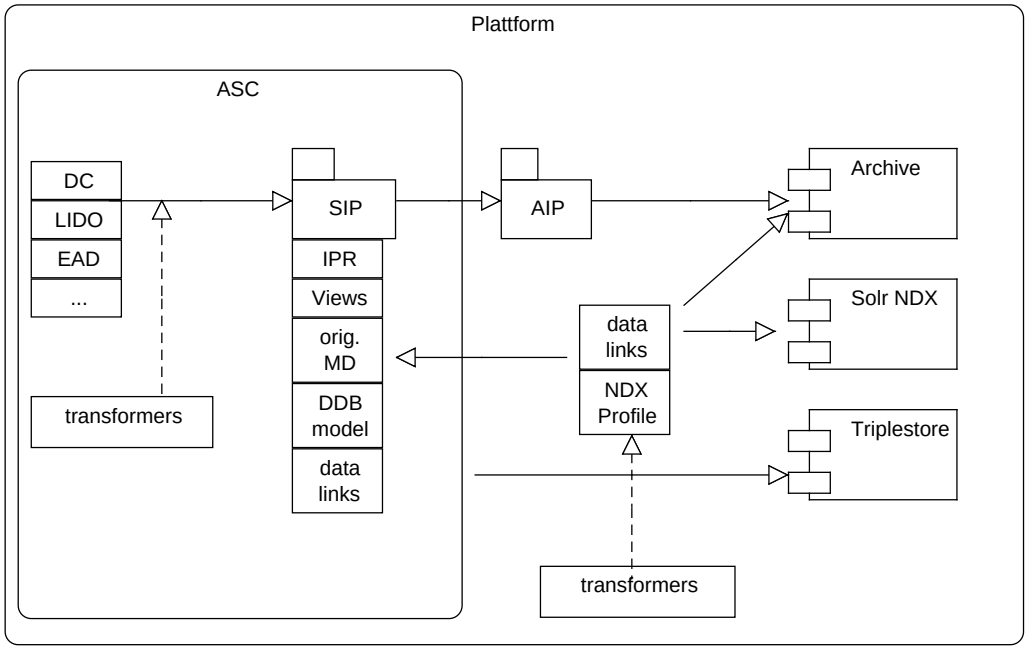

Abbildung 2.2.: Fluss der Metadaten eigene Darstellung, angelehnt an [Su11, S. 15]

[3]Open Archival Information System (OAIS)

3. Grundlagen

Für die Entwicklung eines Editors und Interpreters für eine deklarative Sprache zum Mapping Ontologiemapping ist ein grundlegendes Verständnis der verschiedenen Basistechnologien notwendig. In diesem Abschnitt werden zuerst die Konzepte von Ontologien erläutert, dann die benötigten Techniken.

3.1. Ontologien

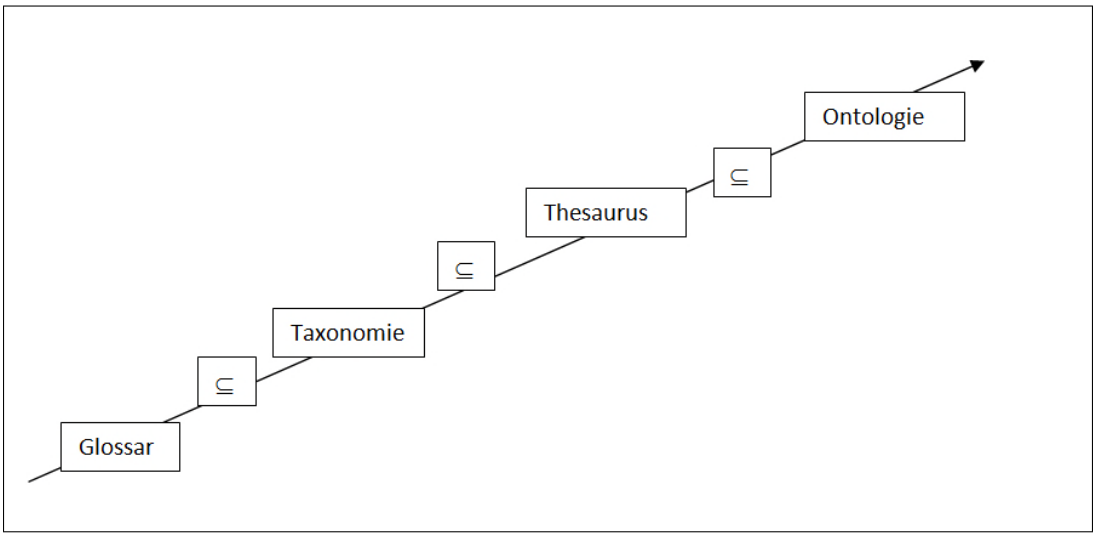

Abbildung 3.1.: Wissensrepräsentationmodell angelehnt an (Abb. [vgl. Ull04])

Um die hohe Menge an Informationen aus den kulturellen Einrichtungen einer breiten Öffentlichkeit zugänglich zu machen, werden für deren Erschließung, Strukturierung und bedarfsgerechte Bereitstellung Ontologien immer wichtiger. Hier ist wegen ihres interdisziplinären Charakters ein strukturiertes Vokabular wichtig (siehe Abb. 3.1).

Durch dem Einsatz von Ontologien wird die Kommunikation erleichtert und transparenter. Gerade im Bereich Wissensaustausch und Informationssysteme ist der Einsatz von Ontologien unverzichtbar. [vgl. BL+05, S. 4 f.]. Im DDB-Projekt wird ein eigenes Model verwendet,

Abbildung 3.2.: Visualisierung eines RDF Beispiels

das auf der CIDOC-CRM Ontologie beruht (siehe Kapitel 3.4). In ihr werden Beziehungen zwischen einem Subjekt (Entity) und Objekt (Entity) durch ein Prädikat (Property) definiert, siehe (Abb. 3.2). Die Suche nach Informationen des kulturellen Erbes soll hiermit beschleunigt und das Ergebnis qualitativ verbessert werden.

3.2. Extensible Markup Language (XML)

XML erfährt eine breite Anwendung, es beruht auf SGML, (SGML, engl., dt. Normierte Verallgemeinerte Auszeichnungssprache). Sie findet Anwendung in verschiedene Auszeichnungssprachen (engl. markup languages) für Dokumente zu definieren, SGML ist Standartisiert in der ISO 8879:1986. SGML und Textformatierung kann auf eine lange Geschichte zurückgreifen, sie stellt eine Unterdisziplin der Computerwissenschaft dar und beruht auf den Veröffentlichungen von Textdokumenten und deren Automatisierung [vgl. GP00, S. 26 f.]. XML Dokumente lassen sich als Baumstruktur repräsentieren (siehe Listing 3.1) und über eine Schnittstelle abfragen. Es gibt zwei Bereitstellungstechniken, sie basieren zum einen SAX (Simple API for XML) und zum anderen DOM (Documenten Object Model) Sie unterscheiden sich nur hinsichtlich der Abfragetechnik [vgl. EE04, S. 7]. In den KWEs werden die Digitalen Objekte (siehe Kapitel 2.6) in XML dargestellt, so lassen sich die Transformationen Regeln mit XSLT (siehe Kapitel 3.5) konstruieren.

3.2.1. Formatierungs-Markup

Markups (Formatierungs-Auszeichnung) werden verwendet um in einem Schriftstück handschriftliche Randbemerkungen anzugeben wie der Text formatiert werden soll. So werden jetzt Dokumente erstellt, die den Text und deren Formatierung beinhalten. Da es keine einheitliche

Markupsprache gab, haben sich viele unterschiedliche Markupsprachen entwickelt. Ende der 60er Jahre gab es Personen, die mehr mit diesen Dokumenten anfangen wollten, bei IBM engagierte man Charles Goldfarb, um für das Verwalten (Speichern, Suchen und Veröffentlichen) von Dokumenten ein System zu entwickeln [vgl. GP00].

3.2.2. Eine allgemeine Dokumentabbildung

Um eine allgemeingültige Dokumentabbildung für alle Programme zu finden, setzen sich Goldfarb, Ed Mosher und Ray Lorie Ende der sechziger Jahre zusammen. Sie erkannten, dass es drei wichtige Schwerpunkte gab. Als erstes musste ein Dokumentenstandard gefunden werden, der alle Programme unterstützt. Denn es ist ersichtlich, wenn nicht alle Programme die gleiche Sprache sprechen, können sie auch nicht zusammen arbeiten [vgl. GP00]. Der zweite Punkt war, dass die Abbildung in eine Standard-Textverarbeitung eine qualitative Weiterverarbeitung möglich macht.

3.2.3. Benutzerdefinierte Dokumenttypen

Damit ein Computersystem mit Text aus ganz verschiedenen Fachgebieten arbeiten kann, muss dem System beigebracht sein, wie es mit verschiedenen Dokumenttypen umgehen soll. Dazu kann eine Deklaration gefunden werden, mit der man mit den Daten arbeiten kann. Die zwar immer gleich ist, sich aber an alle unterschiedlichen Fachgebieten anpassen lässt. Gleichzeitig ist es für die Verarbeitung von Textdokumenten nicht wichtig, wie mit Auszeichnungssprache (Markup) formatiert wird, das kann zwar mit XML gemacht werden, sollte aber vom eigentlichen textuellen Inhalt getrennt werden, dazu eignen sich Stylesheets. Natürlich ist es wichtig, wie ein Dokument angezeigt oder gedruckt wird, niemand möchte XML-Text lesen, aber Markup soll mehr können. Gerade diese Trennung macht die XML-Zukunft sicher, da sollte es neue Darstellungsformen geben, dafür müssen nur die Stylesheets angepasst werden. Markup soll bewirken, Abschnitte des Dokuments zu kommentieren, nicht um sie zu formatieren, sondern um Elementen, wie zum Beispiel dem Verfasser, eine eindeutige Position zu vergeben, um ihnen damit eine logische Rolle zuzuweisen, z.B.: <verfasser>Goethe</verfasser> Sucht man jetzt in einem Dokument nach einem bestimmten Verfasser, so kann man ihn jetzt durch das Markup eindeutig finden, siehe Abb. 3.1.

Listing 3.1: Beispiel Buch

```
<?xml version="1.0">

<buchliste>
<buch>
 <titel>UliStein´sTierleben</titel>
  <autor>
   <vorname>Uli</vorname>
   <nachname>Stein</nachname>
  </autor>
        <buch>
</buchliste>
```

3.2.4. Metadaten

Metadaten sind Daten die wiederum Daten beschreiben, sie stellen eine Informationsressource dar indem sie z.B. ein Buch beschreiben. Solch eine Beschreibung könnte zum Beispiel Informationen über den Verfasser, das Erscheinungsdatum oder den Verlag beinhalten. Mit diesen Daten ist es möglich, semantische Beziehungen zwischen Informationsressourcen darzustellen und eine maschinelle Verarbeitung möglich zu machen. Metadaten kommen in den KWEs schon seit Jahrhunderten zur Anwendung, sind also über lange Zeit gewachsen. Daher gibt es auch unterschiedliche Darstellungsformate (Formatierungs-Markup) siehe Kapitel 3.2.1 [vgl. Pel06, S. 11]. Semantische Beziehungen werden häufig in RDF repräsentiert, das trifft auch in dieser Arbeit zu, siehe Kapitel 3.3.

Listing 3.2: Beispiel Container

```
<?xml version="1.0">

 <buch num="2">
   <author>
    <name>Schmidt</name>
    <vorname>Paul</vorname>
   </author>
   <herausgeber>
    <name>Meier</name>
    <surname>Stefan</surname>
   </herausgeber>
 </buch>
```

3.2.5. Namensräume

Namensräume geben Geltungsbereiche an, sollte es z.B. Namen (s. folgendes Beispiel) in verschiedenen Containerelementen geben, wäre beim Parsen nicht klar definiert, zu welchem Containerelement die beiden Namen gehören würden (siehe Abb. 3.2). Namensräume lösen diese Probleme, indem sie eine Verknüpfung zwischen dem Element und der Quelle der Daten schaffen (siehe Abb. 3.3[vgl. Amm02, S. 35 f.].

Listing 3.3: Beispiel Namespace

```
<?xml version="1.0">
 <buch>
   <author:name>
   <herausgeber:name>
 </buch>
```

3.2.6. Document Object Model (DOM)

Das Document Object Model ist ein Interface zur Bearbeitung von XML Dokumenten, es repräsentiert ein XML Dokument in einer Baumstruktur [vgl. DOM11], in dieser Arbeit benutzen wir den JDOM als Speicher-interne Darstellung von XML [vgl. JDO11]. Hiermit wird ein wahlfreier Zugriff auf einzelne Elemente ermöglicht.

3.3. Resource Description Framework (RDF)

Eine noch mächtigere Auszeichnungssprache als XML ist das Resource Description Framework (RDF), sie findet im DDB-Projekt Einsatz um die Ontologie darzustellen. Mit ihrer Hilfe werden die Beziehungen zwischen Entities repräsentiert (siehe Kapitel 3.4). Suchanfragen lassen sich so durch eine semantische Suche viel gezielter stellen, das Ergebnis wird qualitativ besser [vgl. EE04, S. 236]. RDF ist eine Empfehlung des World Wide Web Consortium (W3C) [vgl. RDF04]. Das RDF-Modell kann grafisch in Form von Knoten und Kanten dargestellt werden (siehe Abb. 3.2). Als Ressourcen werden alle Dinge bezeichnet, sie müssen nur durch eine eindeutige URI[1] gekennzeichnet sein. Entities können Eigenschaften haben, wie zum Beispiel (is title of) , diese werden als Kanten im Graph abgebildet.

3.4. CIDOC Conceptual Reference Model (CIDOC CRM)

Das CIDOC CRM ist eine Ontologie, die vergleichbar einer Hierarchie oder ein objektorientiertes Modell repräsentiert. Sie ist Standard für Museen im kultur- und naturhistorischen Bereich und seit 2006 als ISO Norm (ISO21127) anerkannt [vgl. CID11, S. 7-8]. Als Dokument zur Erläuterung soll die deutsche Übersetzung dienen. Im DDB-Projekt dient das CIDOC CRM als Vorlage für das DDB-eigene Modell und ist daher zum Verständnis des Mappings unbedingt nötig. Es dient als Grundlage zum Informationsaustausch und zur Integration zwischen verschiedenen Informationsquellen wie sie im DDB-Projekt vorkommen. Die Eingangsformate der einzelnen Kultureinrichtungen sind nicht nur verschieden, auch ihre Anwendung ist unterschiedlich. Eine Weiterverarbeitung auf einem System hat zur Folge, sie in ein harmonisiertes Model zu überfuhren. Im DDB-Projekt hat man zum jetzigen Stand das CIDOC CRM als Basis-Model festgelegt [vgl. Su11, S. 49]. Das CRM gibt eine Menge von Begriffen für Objekte (CRM-Klassen) und ihre Eigenschaften (CRM-Properties) vor. Die CRM-Klassen befolgen genau definierte Vererbungsregeln (siehe Abb. 3.5). Auch die Beziehungen (Properties) haben eine Vererbungshierarchie, inklusive Mehrfachvererbung. Eine Ausgangsklasse (Domain) wird durch eine Property an eine Zielklasse (Range) gebunden (siehe Abbildung 3.3). Dabei übernimmt die Domain die Funktion eines Subjekts und der Range die eines Objekts. Hierbei kann jede Klasse eine der beiden Funktionen annehmen. Die Unterscheidung von Domain und Range ist ordinär, denn Properties gibt es meist für beide Richtungen (siehe Abb. 3.3 und 3.4). Die Vererbung ist wie in der objektorientierten Programmierung, eine Unterklasse (subclass) erbt immer die Eigenschaften ihrer Überklasse (superclass), somit ist

[1] Uniform Resource Identifier (URI) (engl. „einheitlicher Bezeichner für Ressourcen“) [vgl. Bon08, S. 1120]

die subclass immer eine Spezialisierung und die superclass immer eine Generalisierung einer oder mehrerer Klassen siehe [vgl. CID11, S. 22 f.]. Es gibt 90 Klassen und 148 Eigenschaften [siehe CID11, S. 42 f.,117 ff.].

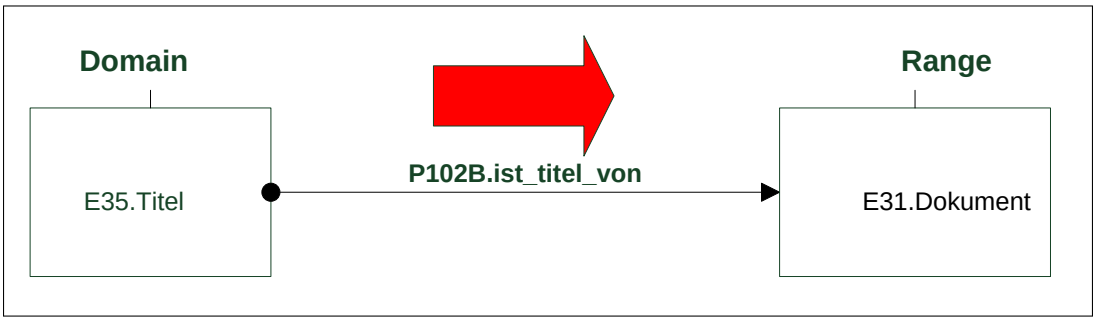

Abbildung 3.3.: Domain Range (B)

Abbildung 3.4.: Domain Range (F)

3.5. Extensible Stylesheet Language (XSL)

Metadaten Objekte sind in der DDB-Projekt in XML beschreiben. Um sie in das DDB-Projekt zu importieren, müssen sie von einem Quellmetadatenformat in ein Zielmetadatenformat überführt werden. Die Quellmetadaten werden dabei nicht verändert. Die Inhalte für die Zielmetadaten können aus dem Quellmetadaten-Objekt stammen oder durch Rekombinieren ermittelt werden. Der Import der DDB-Daten, wie sie in dieser Arbeit betrachtet werden, wird mittels XSL (Extensible Stylesheet Language) beschrieben.

XSL zerfällt in zwei Teilbereiche, zum einen in den Präsentations-Bereich, der andere beschreibt ausschließlich den Transformations-Prozess. Diese Arbeit wird sich vorwiegend mit dem Transformationsprozess beschäftigen. Zur Transformation werden Extensible Stylesheet

Entities		Vererbungsebenen						
E1	CRM Entität							
E2	-	Geschehendes						
E4	-	-	Phase					
E5	-	-		Ereignis				
E7	-	-	-	-	Handlung			
E11	-	-	-	-	-	Bearbeitung		
E12	-	-	-	-	-	-	Herstellung	
E13	-	-	-	-	-	Merkmalszuweisung		
E65	-	-	-	-	-	Begriffl iche Schöpfung		
E63	-	-	-	-	Daseinsbeginn			
E12	-	-	-	-	-	Herstellung		
E65	-	-	-	-	-	Begriffl iche Schöpfung		
E64	-	-	-	-	Daseinsende			
E77	-	Seiendes						
E70	-	-	Sache					
E72	-	-	-	Rechtsobjekt				
E18	-	-	-	-	Materielles			
E24	-	-	-	-	-	Hergestelltes		
E90	-	-	-	-	Sinnbild			
E71	-	-	-	Künstliches				
E24	-	-	-	-	Hergestelltes			
E28	-	-	-	-	Begrifflicher Gegenstand			
E89	-	-	-	-	-	Aussagenobjekt		
E30	-	-	-	-	-	-	Recht	
E73	-	-	-	-	-	-	Informationsgegenstand	
E90	-	-	-	-	-	Sinnbild		
E41	-	-	-	-	-	-	Benennung	
E73	-	-	-	-	-	-	Informationsgegenstand	
E55	-	-	-	-	-	Typus		
E39	-	-	Akteur					
E74	-	-	-	Menschliche Gruppe				
E52	-	Zeitspanne						
E53	-	Ort						
E54	-	Maß						
E59	Primitiver Wert							
E61	-	Zeitprimitiv						
E62	-	Zeichenkette						

Abbildung 3.5.: Auszug aus der Entities Vererbungsebenen angelehnt an [Abb. CID11, S. 15]

ID	Name der Eigenschaft	Ausgangsklasse	Zielklasse
P1	wird bezeichnet als (bezeichnet)	E1 CRM Entität	E41 Benennung
P2	hat den Typus (ist Typus von)	E1 CRM Entität	E55 Typus
P3	hat Anmerkung	E1 CRM Entität	E62 Zeichenkette
P4	hat Zeitspanne (ist Zeitspanne von)	E2 Geschehendes	E52 Zeitspanne
P7	fand statt in (bezeugte)	E4 Phase	E53 Ort
P10	fällt in (enthält)	E4 Phase	E4 Phase
P12	fand statt im Beisein von (war anwesend bei)	E5 Ereignis	E77 Seiendes
P11	hatte Teilnehmer (nahm Teil an)	E5 Ereignis	E39 Akteur
P14	wurde ausgeführt von (führte aus)	E7 Handlung	E39 Akteur
P16	benutzte das bestimmte Objekt (wurde benutzt für)	E7 Handlung	E70 Sache
P31	veränderte (wurde verändert durch)	E11 Bearbeitung	E24 Hergestelltes
P108	hat hergestellt (wurde hergestellt durch)	E12 Herstellung	E24 Hergestelltes
P92	brachte in Existenz (wurde in Existenz gebracht durch)	E63 Daseinsbeginn	E77 Seiendes
P108	hat hergestellt (wurde hergestellt durch)	E12 Herstellung	E24 Hergestelltes
P94	hat erschaffen (wurde erschaffen durch)	E65 Begriffliche Schöpfung	E28 Begriffl icher Gegenstand
P93	beendete die Existenz von (wurde seiner Existenz beraubt durch)	E64 Daseinsende	E77 Seiendes
P15	wurde beeinfl usst durch (beeinflusste)	E7 Handlung	E1 CRM Entität
P16	benutzte das bestimmte Objekt (wurde benutzt für)	E7 Handlung	E70 Sache
P20	hatte den bestimmten Zweck (war Zweck von)	E7 Handlung	E7 Handlung
P43	hat Dimension (ist Dimension von)	E70 Sache	E54 Maß
P46	ist zusammengesetzt aus (bildet Teil von)	E18 Materielles	E18 Materielles
P59	hat Bereich (befi ndet sich auf oder in)	E18 Materielles	E53 Ort
P67	verweist auf (wird angeführt von)	E89 Aussagenobjekt	E1 CRM Entität
P75	besitzt (sind im Besitz von)	E39 Akteur	E30 Recht
P81	andauernd während	E52 Zeitspanne	E61 Zeitprimitiv
P82	irgendwann innerhalb von	E52 Zeitspanne	E61 Zeitprimitiv
P89	fällt in (enthält)	E53 Ort	E53 Ort
P104	Gegenstand von (fi ndet Anwendung auf)	E72 Rechtsobjekt	E30 Recht
P106	ist zusammengesetzt aus (bildet Teil von)	E90 Sinnbild	E90 Sinnbild
P107	hat derzeitiges oder früheres Mitglied (ist derzeitiges oder früheres Mitglied von)	E74 Menschliche Gruppe	E39 Akteur
P127	hat den Oberbegriff (hat den Unterbegriff)	E55 Typus	E55 Typus
P128	trägt (wird getragen von)	E24 Hergestelltes	E73 Informationsgegenstand
P130	zeigt Merkmale von (Merkmale auch auf)	E70 Sache	E70 Sache
P140	wies Merkmal zu (bekam Merkmal zugewiesen durch)	E13 Merkmalszuweisung	E1 CRM Entität
P141	wies zu (wurde zugewiesen durch)	E13 Merkmalszuweisung	E1 CRM Entität
P148	hat Bestandteil (ist Bestandteil von)	E89 Aussagenobjekt	E89 Aussagenobjekt

Abbildung 3.6.: Auszug aus der Properties to Entities angelehnt an [Abb. CID11, S. 16]

Language „Transformations“ (XSLT) formuliert. Neben XSLT wird auch die XML Path Language (XPath)-Syntax verwendet. XPath ermöglicht den gezielten Zugriff auf den XML-Inhalt [vgl. Bon08, S. 27 ff.].

4. Augmented SIP Creator (ASC)

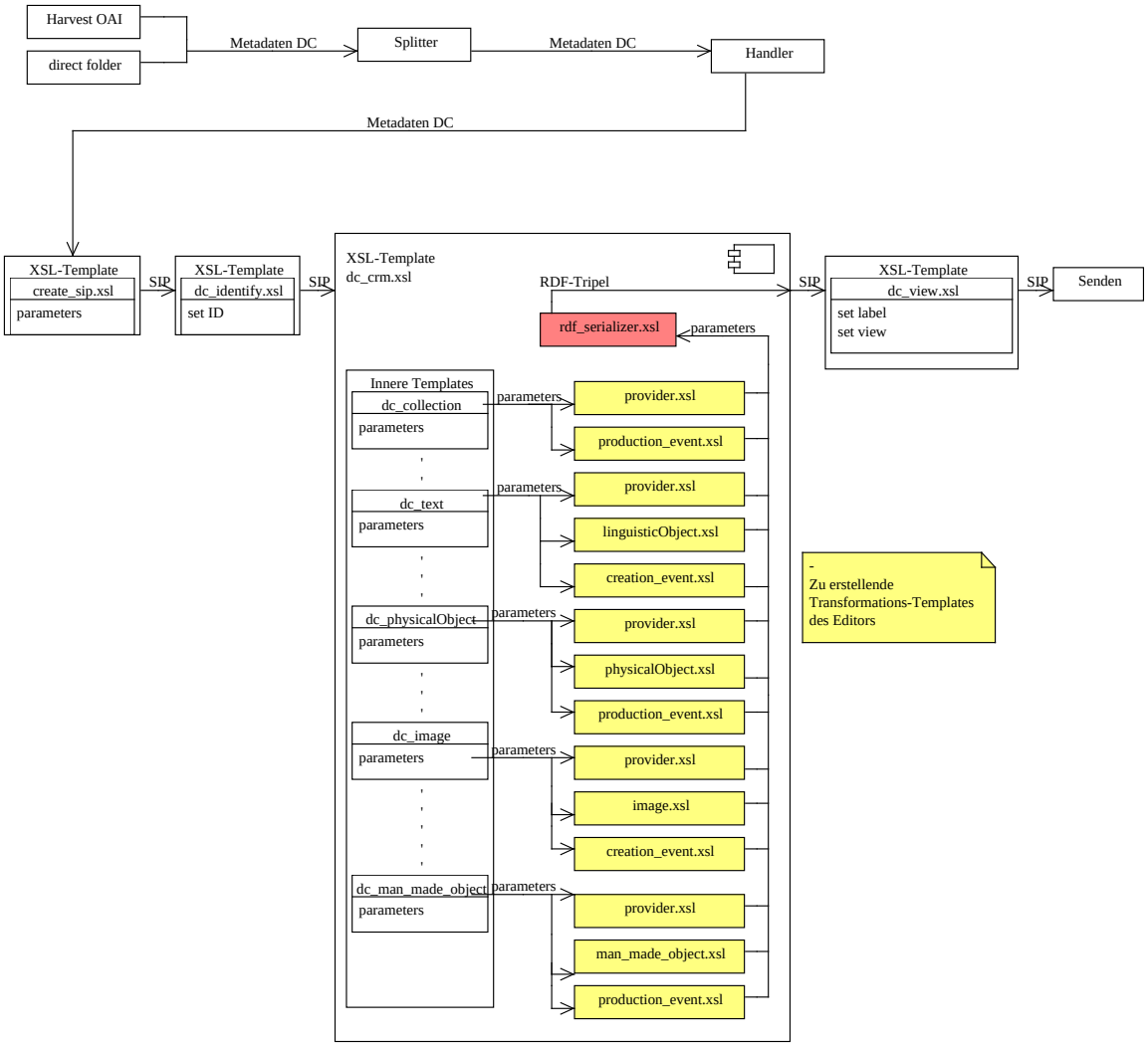

Abbildung 4.1.: Transformationsprozess am Beispiel von DC Metadaten

In diesem Kapitel wird der ASC als externe Komponente des DDB-Projekts erläutert, dies ist zum Verständnis dieser Arbeit wichtig, um den Fluss der Daten sowie deren Transformation

nachvollziehen zu können.

Abbildung 4.1 zeigt eine Übersicht über den Transformationsprozess des ASC am Beispiel von DC-Metadaten. Die Dublin-Core-Daten (DC)-Daten werden entweder über eine OAI-Schnittstelle heruntergeladen oder aus einem bestehenden Verzeichnis importiert. Jede importierte Datei kann mehrere Objekte enthalten; diese werden von einem Splitter aufgeteilt. Der Handler eliminiert alle nicht benötigten Sonderzeichen aus den Metadaten. Zum Ingestieren der Daten dient das vereinheitliche DDB-Datenformat SIP[1] .

4.1. SIPs

Das Erzeugen eines Session Initiation Protocol (SIP) ist in mehrere Transformationsprozesse untergliedert. Die Prozesse laufen sequentiell ab, dabei strukturieren die horizontal dargestellten Prozesse das SIP und versorgen die vertikal dargestellten mit den Inhalten der Metadaten als Parameter. Der vertikale Prozess transferiert diese Parameter, als unterste Ebene erzeugt der RDF-Serializer RDF-Tripel und fügt sie dem SIP zu.

Horizontal dargestellter Transformationsprozess:

1. dc_create_sip.xsl
    - erzeugt für jedes Objekt ein SIP.
    - übernimmt die Metadaten.
2. dc_identify.xsl
    - ermittelt aus dem Metadaten über einer Prioritätsliste die ID
3. dc_crm.xsl
    - interner Transformationprozess
        a) dc_collection
        b) dc_text
        c) dc_physicalObject
        d) dc_image
        e) dc_man_made_object
4. dc_views.xsl

[1] „Submission Information Package“, eine Bezeichnung der OAIS

- setzen des HTML Text für die Suche
- setzen des HTML Text für die View

Vertikal dargestellter Transformationsprozess:

1. dc_collection
    - provider.xsl rdf_serializer.xsl
    - production_event.xsl rdf_serializer.xsl
2. dc_text
    - provider.xsl rdf_serializer.xsl
    - liguisticObject.xsl rdf_serializer.xsl
    - creation_event.xsl rdf_serializer.xsl
3. dc_physicalObjekt
    - provider.xsl rdf_serializer.xsl
    - physicalObject.xsl rdf_serializer.xsl
    - production_event.xsl rdf_serializer.xsl
4. dc_man_made_object
    - provider.xsl rdf_serializer.xsl
    - man_made_object.xsl rdf_serializer.xsl
    - production_event.xsl rdf_serializer.xsl

4.2. RDF Transformation

Wie im Kapitel 4.1 erläutert, führt der vertikal dargestellte Prozess zum Erzeugen eines SIP, dieser Prozess ist für das Verständnis des RDF-XSLT-Editors maßgeblich von Bedeutung. Dieser Prozess harmonisiert die unterschiedlichen Metadaten auf ein einziges Format (CIDOC-CRM) und setzt ihre Inhalte zueinander in Beziehung. Als Beispiel soll das DC-Metadatenformat dienen, siehe Abbildung (4.2). Wie in der Abbildung dargestellt, entsteht bei der Transformation ein Graph, jede aus den DC-Daten sinnvoll ableitbare Beziehung im CIDOC CRM wird in ein RDF-Tripel abgebildet. Jedes SIP spiegelt für sich selbst einen Graphen wieder,

mit meist mehreren Beziehungen, alle SIPs bilden einen Gesamtgraphen. Dieser Graph stellt die gesamte Wissensrepräsentation dar, dieser wird am Ende ca. 300 Millionen SIPs mit wiederum einer Vielzahl an Beziehungen enthalten.

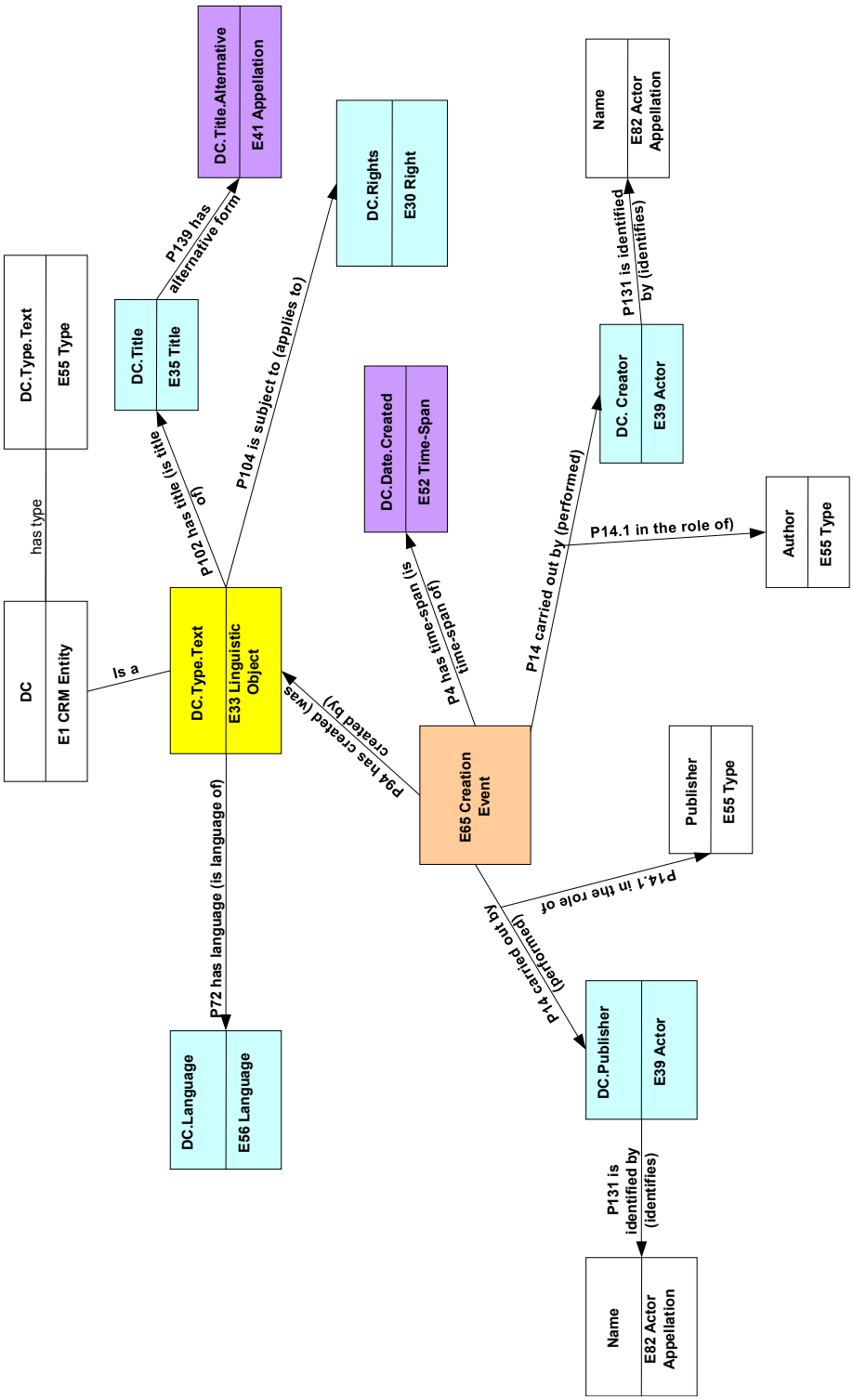

Abbildung 4.2.: Transformationmodell DC Mapping nach DCCRM [siehe Su11, S. 8]

5. Konzeption des RDF-XSLT-Editors

Im Zuge dieser Abschlussarbeit wurde ein Editor für die Produktion von Transformations-Templates zur Produktion semantischer Relationen entwickelt. Als Programmiersprache wurde Java 6 gewählt, als Entwicklungsumgebung Eclipse[1] in der Version 3.6, um zum DDB-Projekt kompatibel zu sein. Die grafische Oberfläche wurde mit AWT[2] und Swing[3] implementiert. Für die Darstellung des Graphen wurde das Framework JGraph eingesetzt.

5.1. Einführung in die technische Konzeption

Im diesen Kapitel wird die technische Konzeption des Editors an einem allgemein verständlichen Transformationsprozess erläutert und ein Überblick über die Funktionsweise gegeben. Als Ausgangspunkt dient ein DC-Metadatensatz von "Harry Potter und der Halbblutprinz"von Joanne K. Rowling, aus diesem werden nur die zur Erläuterung wichtigen Teile herausgenommen (siehe Kapitel 5.1).

[1] Eclipse ist eine integrierte Entwicklung (IDE) der Eclipse Foundation

[2] Abstract Window Toolkit (AWT) ist eine API zur Entwicklung von grafischen Oberflächen. Sie ist in Java [vgl. BP08, S. 13]

[3] Swing ist ein Rahmenwerk(framework) und baut zu Teilen auf AWT auf und erweitert dieses [vgl. BP08, S. 13]

Listing 5.1: DC-Metadaten

```
<?xml version="1.0">
<item>
 <dc:title>Harry Potter und der Halbblutprinz</dc:title>
 <dc:author>Joanne K. Rowling</dc:author>
 <!-- ... -->
</item>
```

Der Transformationsprozess hat mehrere Hierarchie-Ebenen. Die erste Ebene bereitet den Transformationsprozess vor, indem sie die Informationen in Parameter transferiert (siehe Kapitel 4.1). Darauf folgen weitere Transformations-Templates, dabei ist das letzte Transformations-Template immer der RDF-Serializer. Das Transformations-Template, das den RDF-Serializer aufruft, wird von dem in dieser Arbeit entwickelten Editor produziert (siehe Listing 5.3). Der eigentliche Aufruf des RDF-Serializers ist für eine genauere Betrachtung nochmals in Listing 5.2 wiedergegeben.

Listing 5.2: Aufruf des RDF-Serializers

```
<xsl:call-template name="rdf_serializer">
  <xsl:with-param name="crm_property" select=
          "'crm:P102B.is_title_of'"/>
        <xsl:with-param name="range_type" select=
                "'http://www.cidoc-crm.org/crm-concepts/#E31'"/>
  <xsl:with-param name="value" select="\$title"/>
        <xsl:with-param name="domain_id" select="\$local_id"/>
  <xsl:with-param name="domain_type" select=
          "'http://www.cidoc-crm.org/crm-concepts/#E35'"/>
        <xsl:with-param name="datatype" select="coverage"/>
  <xsl:with-param name="nexus_id" select="source"/>
</xsl:call-template>
```

Für jede Beziehung zwischen zwei Entities muss einmal mit `xsl:call-template name="rdf_serializer"` der Template-Serializer aufgerufen werden, dieses Template produziert dann RDF-Tripel und trägt diese in das SIP ein. Nach diesem Prinzip funktionieren alle XSLTs vor dem `rdf_serializer.xsl` Template in Abbildung 4.1.

5.2. Technische Konzeption

Die Architektur des Editors gliedert sich in die Komponenten „User Interface“, „Model“ und „Ontology“. Durch diese Komponenten wird ein Schichtenmodell definiert, siehe Abbildung

5.1. Das Paket „User Interface“ umfasst die Benutzeroberfläche des Editors, die in mehrere Bereiche gegliedert ist. Diese bilden die einzelnen Funktionen (Editieren der Namespaces, der Parameter und des Graphen) ab. Das Framework JGraph wird zur Erstellung des RDF-Graphen verwendet, mit dessen Hilfe in einem weiteren Schritt die XSLT-Skripte und damit das Mapping der Eingangsformate auf das DDB-eigene Modell erzeugt werden. Hierzu können Knoten und Kanten unter Berücksichtigung des DDB-Modells gesetzt und verbunden werden, siehe Kapitel 3.4. Die Komponente „Model“ verwaltet die Veränderungen der Parameter zur Laufzeit, für diese Parameter werden im letzten Schritt die Transformations-Templates erstellt. Die unterste Schicht „Ontology“ unterstützt den Anwender, indem sie berechnet, welche Beziehungen zwischen den Entities erzeugt werden dürfen.

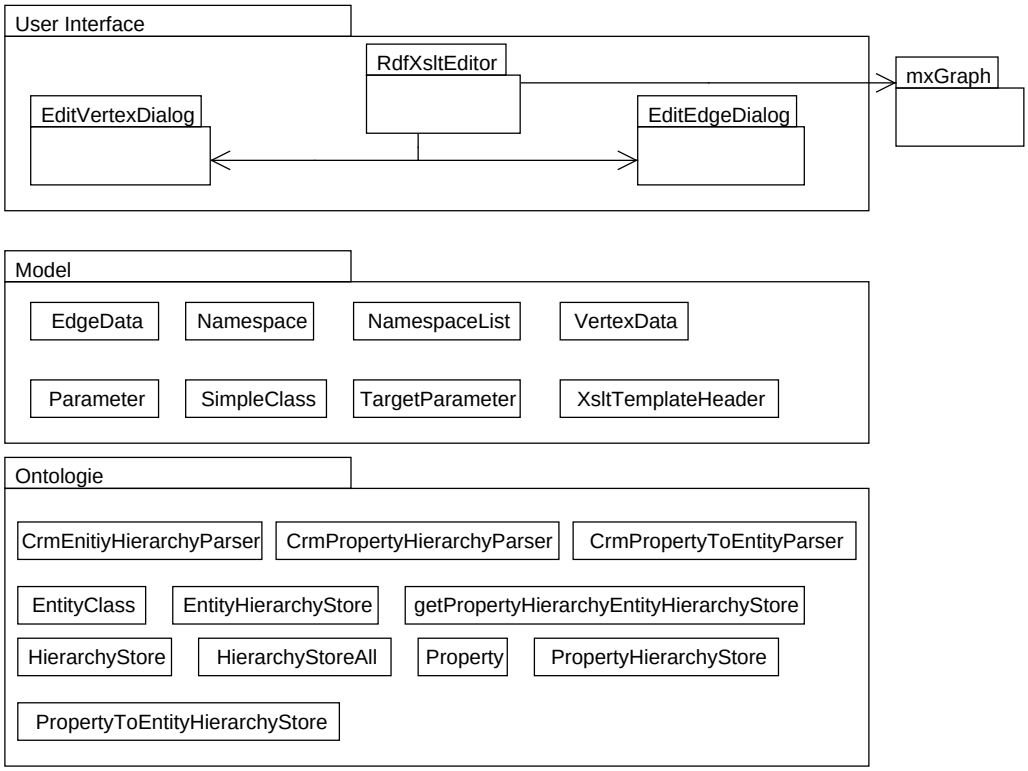

Abbildung 5.1.: Schichtenarchitektur des Editors

5.3. Technische Umsetzung

Listing 5.3: Transformations-Template (komplett)

```
<!--XSLT Template-->

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsl:stylesheet xmlns:crm="http://www.cidoc-crm.org/crm/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="2.0">

  <xsl:template name="XSLT Template">
    <xsl:with-param name="local_id"/>
    <xsl:with-param name="title"/>
    <xsl:with-param name="type"/>

    <xsl:call-template name="rdf_serializer">
      <xsl:with-param name="crm_property" select=
          "'crm:P102B.is_title_of'"/>
      <xsl:with-param name="range_type" select=
          "'http://www.cidoc-crm.org/crm-concepts/#E31'"/>
      <xsl:with-param name="value" select="\$title"/>
      <xsl:with-param name="domain_id" select="\$local_id"/>
      <xsl:with-param name="domain_type" select=
          "'http://www.cidoc-crm.org/crm-concepts/#E35'"/>
      <xsl:with-param name="datatype" select="\$type"/>
      <xsl:with-param name="nexus_id" select="\$local_id"/>
    </xsl:call-template>
  </xsl:template>
</xsl:stylesheet>
```

Als Muster für die Konzeption der technischen Umsetzung wurde ein handkodiertes XSL-Transformations-Template ähnlich zu Listing 5.3 verwendet, welches das gewünschte Ergebnis des Editors darstellt. Aus diesem Muster kann man die Struktur der zu erzeugenden XSLT-Skripte ableiten: Der erste Teilbereich beginnt mit der Deklaration für XML- Dokumente (siehe Kapitel 3.2), gefolgt von den Namensräumen (siehe Kapitel 3.3), die im Editor durch Objekte repräsentiert werden (siehe Abbildung 5.1). Im zweiten Teilbereich werden die Parameter definiert, die im Editor ebenfalls durch Objekte repräsentiert werden. Der dritte Teil ruft das Template „RDF-Serializer“ auf und setzt die Werte wie in Abbildung 5.3. Das RDF-Serializer-Template ist in der Hierarchie das letzte Template und befüllt mit den RDF-Tripeln das SIP, siehe Kapitel 4.1.

Zur technischen Umsetzung der CIDOC CRM Ontologie wird die RDF-Repräsentation gewählt, mit ihr werden Klassen (Entities) (siehe Abbildung 3.5) mit der Eigenschaft (Properties) (siehe Abbildung 3.6) in Beziehung gesetzt. Diese lässt sich im einem XML-Dokument `cidoc_crm_label.xml` mit RDF Repräsentation formulieren, siehe Beispiele 5.4 und 5.5. Dieses Dokument wird als Quelle zur Berechnung der Klassenhierarchie verwendet (siehe Abbildung 3.3). Dies ist relevant, weil Entities auch mit Properties ihrer Oberklassen verbunden sein dürfen.

Listing 5.4: cidoc_crm_label.xml Ausschnitt der Entities

```
<rdfs:Class rdf:ID="E1.CRM_Entity">
 <rdfs:comment>
 <!-- Beschreibung [siehe CID11] -->
 </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="E2.Temporal_Entity">
  <rdfs:comment>
   <!-- Beschreibung [siehe CID11] -->
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#E1.CRM_Entity"/>
</rdfs:Class>

<rdfs:Class rdf:ID="E12.Production">
 <rdfs:comment>
  <!-- Beschreibung [siehe CID11] -->
 </rdfs:comment>
 <rdfs:subClassOf rdf:resource="#E11.Modification"/>
 <rdfs:subClassOf rdf:resource="#E63.Beginning_of_Existence"/>
</rdfs:Class>
</rdf>
```

Listing 5.5: cidoc_crm_label.xml Ausschnitt der Properties

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rdf:RDF xml:lang="en" xmlns:rdfs=
  "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <rdf:Property rdf:ID="P1F.is_identified_by">
  <rdfs:comment>
    <!-- Beschreibung [siehe CID11] -->
  </rdfs:comment>
  <rdfs:domain rdf:resource="#E1.CRM_Entity"/>
  <rdfs:range rdf:resource="#E41.Appellation"/>
 </rdf:Property>
 <rdf:Property rdf:ID="P1B.identifies">
  <rdfs:domain rdf:resource="#E41.Appellation"/>
  <rdfs:range rdf:resource="#E1.CRM_Entity"/>
 </rdf:Property>
<rdf:Property rdf:ID="P2F.has_type">
  <rdfs:comment>
   <!-- Beschreibung [siehe CID11] -->
  </rdfs:comment>
  <rdfs:domain rdf:resource="#E1.CRM_Entity"/>
  <rdfs:range rdf:resource="#E55.Type"/>
 </rdf:Property>
 <rdf:Property rdf:ID="P2B.is_type_of">
  <rdfs:domain rdf:resource="#E55.Type"/>
  <rdfs:range rdf:resource="#E1.CRM_Entity"/>
 </rdf:Property>
 <rdf:Property rdf:ID="P16F.used_specific_object">
  <rdfs:comment>
   <!-- Beschreibung [siehe CID11] -->
  </rdfs:comment>
  <rdfs:domain rdf:resource="#E7.Activity"/>
  <rdfs:range rdf:resource="#E70.Thing"/>
  <rdfs:subPropertyOf rdf:resource=
           "#P12F.occurred_in_the_presence_of"/>
  <rdfs:subPropertyOf rdf:resource="#P15F.was_influenced_by"/>
 </rdf:Property>
 <rdf:Property rdf:ID="P16B.was_used_for">
  <rdfs:domain rdf:resource="#E70.Thing"/>
  <rdfs:range rdf:resource="#E7.Activity"/>
  <rdfs:subPropertyOf rdf:resource="#P12B.was_present_at"/>
  <rdfs:subPropertyOf rdf:resource="#P15B.influenced"/>
 </rdf:Property>
</rdf>
```

Für den Zugriff auf alle Entities und ihre jeweiligen Properties gab es zuerst den Ansatz, das `cidoc_crm_label.xml` Dokument in einen Jena RDF-Triple-Store[4] zu laden, um dann durch SPARQL-Anfragen die benötigten Ergebnisse zu bekommen. Dieser Ansatz wurde aber während dieser Arbeit aus Performance-Gründen verworfen.

Der neue Ansatz zur Berechnung liest das `cidoc_crm_label.xml` Datei als JDOM Document ein (siehe 3.2.6), danach lassen sich mit unterschiedlichen Parsern aus dem Dokument alle Entities (Knoten), Properties (Kanten) und ihre Beziehungen berechnen. Diese werden benötigt, damit ein Anwender bei der Konstruktion eines Graphen die Zuweisungen der Knoten sowie der Kantenbezeichnungen vornehmen kann und dabei die Regeln der Ontologie einhält (siehe Kapitel 3.4). Weiterhin soll es möglich sein, die Ontologie nachträglich zu ändern oder gänzlich zu ersetzen. Diese Art der Umsetzung stellt nur minimale Anforderungen an eine Ontologie, sie muss sich nur im RDF Syntax darstellen lassen.

Das Parsen der Hierarchie ist im Editor im Paket XHierarchie implementiert, wie im Kapitel 3.2.6 beschrieben ist, zuerst wird eine Document doc initialisiert und mit dem `cidoc_crm_la bel.xml` Dokument geladen. Mit einem Iterator hat man dann einen Zugriff auf alle Elemente des Dokuments. Die Parser werden hier so beschrieben, dass sie alle Unterklassen berechnen. Für die Berechnung von bestimmten Klassen und Unterklassen sind nur wenige Modifikationen vorzunehmen, die hier nicht gesondert vorgestellt werden.

5.3.1. Parsen der Entity-Hierarchie

Abbildung 5.4 zeigt, wie jede Entity mit ihren Unterklassen angegeben wird, dabei ist die Unterklasse aus der Sicht von E2 zu sehen: E2 ist Unterklasse von E1, also ist E1 die Superklasse von E2.

Das Parsen aller Entites aus dem `cidoc_crm_label.xml` Dokument ist über mehrere Klassen bzw. ihre Methoden verteilt (siehe Abb. 5.2). Mit dem Aufruf aus dem User Interface werden folgende Methoden aufgerufen:

- `HierarchyStoreAll`
  Das User Interface ruft die Methode `getEntityHierarchyAll()` auf, um als Ergebnis ein String-Array mit allen Entities zu erhalten. Dazu wird zunächst durch den Aufruf der Methode `getAllSubClasses(String entity)` aus der Klasse `Ent`

[4] Jena ist ein Semantic Web Framework mit einfacher RDFS und OWL-Lite Reasoner. Jena entstand aus dem Hewlett-Packard Semantic Web Research Programm und wurde als Open Source Software auf SourceForge veröffentlicht [vgl. Jen11]

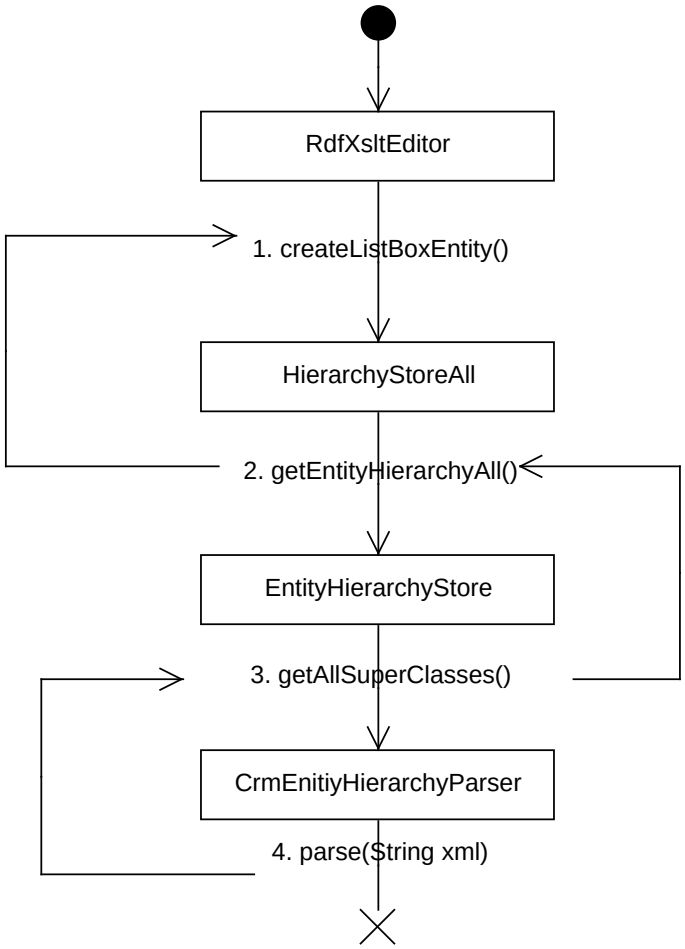

Abbildung 5.2.: Parsen der Entity-Hierarchie

`ityHier archyStore` mit der obersten Entity-Klasse `E1.CRM_Entity` als Parameter ein Set aller Entities erzeugt, das dann in ein Array umgeformt wird. Da das Set für jede Entity noch eine URI mitführt, wird diese bei der Umwandlung in ein Array abgespalten, so dass in dem Array nur noch die Bezeichnungen der Entity ist.

- `EntityHierarchyStore`
  Die Methode `getAllSubClasses()` bekommt einen String übergeben, dieser besteht aus einen URI und einer Entity. Die übergebene Entity bestimmt, welche Entities zurückgegeben werden, das berechnet sich aus ihrer Position in der Hierarchieebene, siehe Abbildung 3.5. Es werden alle Entities zurückgegeben, die unterhalb ihrer Hierarchieebene liegen, plus der benannten. Um diese zu bekommen, übergibt sie

ein neues String Set `entityClass` und den zuvor erhaltenen String an die Methode `determineSubCla ssesRecursive()"`. Diese Methode füllt rekursiv das Set, in dem sie einmal die Methode `init()` aufruft. Die Methode `init()` weist dem neu erstellten String „xml“ das `cidoc_crm_label.xml` zu und ruft damit die Methode `parse(xml)` aus der Klasse `CrmEntityHierarchyParser` auf, die ihren Rückgabewert der Klassenmap `entityClassMap` zuweist. Die Methode `determineSubClassesRecursive()` strukturiert die Klassenmap `entityCla ssMap`, die jetzt alle Entities mit ihren jeweiligen Superklassen beinhaltet, so um, dass alle Entities mit ihren Unterklassen in das Set `entityClass` gespeichert werden können. Dieses Set übergibt dann die Methode `getAllSubClasses()`.

- `CrmEntityHierarchyParser`
  aus ihr wird, wie vorher schon beschrieben, die Methode `parse(String xml)` aufgerufen, die zunächst den xml String in ein Document-Objekt umwandelt (wie im Kapitel 5.3 schon erläutert). Dann wird eine Map erzeugt, die für jede Entity ein entsprechendes EntityClass-Objekt enthält. Dies geschieht in der Methode `createEntityMap (Document doc)`, die zunächst eine leere Map erzeugt.Diese wird dann durch wiederholtes Aufrufen der Methode `putIntoMap(elem, map)` für jedes Element des XML-Dokuments befüllt.

  Die Methode `putIntoMap(elem, map)` ermittelt zunächst den vollständigen Namen der Entity und erzeugt dann ein neues EntityClass-Objekt. Diesem EntityClass-Objekt werden dann alle Superklassen der Entity hinzugefügt. Zum Schluss wird der übergebenen Map der neue Eintrag (Mapping des Entity-Namens auf das soeben erzeugte EntityClass-Objekt) hinzugefügt.

5.3.2. Parsen der Property-Hierarchie

Auch für die Properties gilt wie im Abbildung 5.3 zu sehen ist, dass Properties Unterklassen haben können, dabei ist Unterklasse aus der Sicht von P2 zu sehen: P2 ist Unterklasse von P1 also ist P1 die Superklasse von P2.

Das Parsen aller Properties aus dem `cidoc_crm_label.xml`-Dokument ist über mehrere Klassen und ihre Methoden verteilt (siehe Abbildung 5.3). Mit dem Aufruf aus dem User Interface werden dann folgende Methoden aufgerufen:

- `HierarchyStoreAll`
  Das User Interface ruft die Methode `getPropertyHierarchyAll()` auf, um als

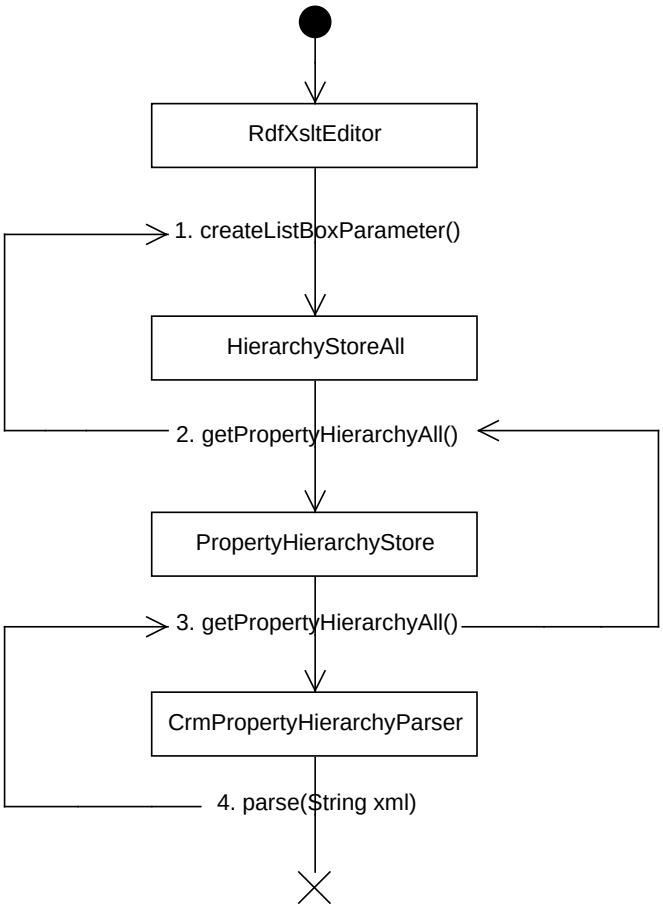

Abbildung 5.3.: Parsen der Properties-Hierarchie

Ergebnis ein String Array mit allen Properties zu erhalten. Die Methode erstellt das String Array aus einem Set, das sie durch den Aufruf von der Methode `getPropertyHierarchyAll()` aus der Klasse `PropertyHierarchyStore` bekommt. Dazu nimmt sie aus dem Set für jede Property einen String, dieser hat noch eine URI vor der Property Bezeichnung, ein Splitter teilt den String so, dass im dem Array nur noch die Bezeichnung der Property ist.

- `PropertyHierarchyStore`
  Die Methode `getPropertyHierarchyAll()` erstellt eine neue ArrayList `results2`, befüllt sie, indem sie eine neue Map `results3` erstellt und ihr den Inhalt von der Klassenmap `propertyMap` zuweist, von jeden Element speichert sie den Key in `results2` und übergibt es der aufrufenden Methode. Bei der Initialisierung der Klasse

PropertyHierarchyStore ruft der Konstrukteur die Methode init() auf. Die Methode init() weist dem neu erstellten String „xml“ das cidoc_crm_label.xml zu und ruft damit die Methode propertyHierarchyParser.parse(xml) aus der Klasse CrmPropertyHierarchyParser auf, die ihren Rückgabewert der Klassenmap propertyMap zuweist.

- CrmPropertyHierarchyParser
  aus ihr wird, wie vorher schon beschrieben, die Methode parse(String xml) aufgerufen, die zunächst den xml String in ein Document-Objekt umwandelt (wie im Kapitel 5.3 schon erläutert). Dann wird eine Map erzeugt, die für jede Property ein entsprechendes Property-Element enthält, die zunächst eine leere Map erzeugt und diese dann durch wiederholtes Aufrufen der putInto(propertyElem, result) für jedes Element des XML-Dokuments befüllt.

  Die Methode putInto(propertyElem, result) ermittelt zunächst den vollständigen Namen der Property indem sie die Methode getPropertyName(Element elem) aufruft und erzeugt dann ein neues Property-Objekt. Diesem Property-Objekt werden dann alle Superklassen der Property hinzugefügt. Zum Schluss wird der übergebenen Map der neue Eintrag (Mapping des Property-Namens auf das soeben erzeugte Property-Objekt) hinzugefügt.

5.3.3. Bestimmen der Zuordnung von Properties zu Entities

Um die zulässigen Properties zu einer Domain zu bestimmen, müssen zunächst alle Sub-Entities zu dieser Domain berechnet werden (siehe Kapitel 5.3.1). Zu diesen Entities werden jetzt alle Properties berechnet (siehe Abb. 5.4). Zum Schluss müssen noch für diese Properties alle Sub-Properties ermittelt und hinzugefügt werden.

5.3.4. Produktion des Transformations-Templates

Die Klasse RdfXsltEditor ist für die Produktion des Transformations-Templates verantwortlich (siehe Abbildung 5.5), hierzu initialisiert sie eine Instanz der Klasse GraphModelData, welche zwei Methoden aufruft: Zum einen die Methode initNamespaceList(), die die vordefinierten Namespace-Werte in einer Klasse NamespaceList speichert, zum anderen die Methode initFactureParameter(), die zu Demonstrationszwecken ein Objekt der Klasse ParameterListe mit vordefinierten Parametern füllt. Die Klasse EdgeDa

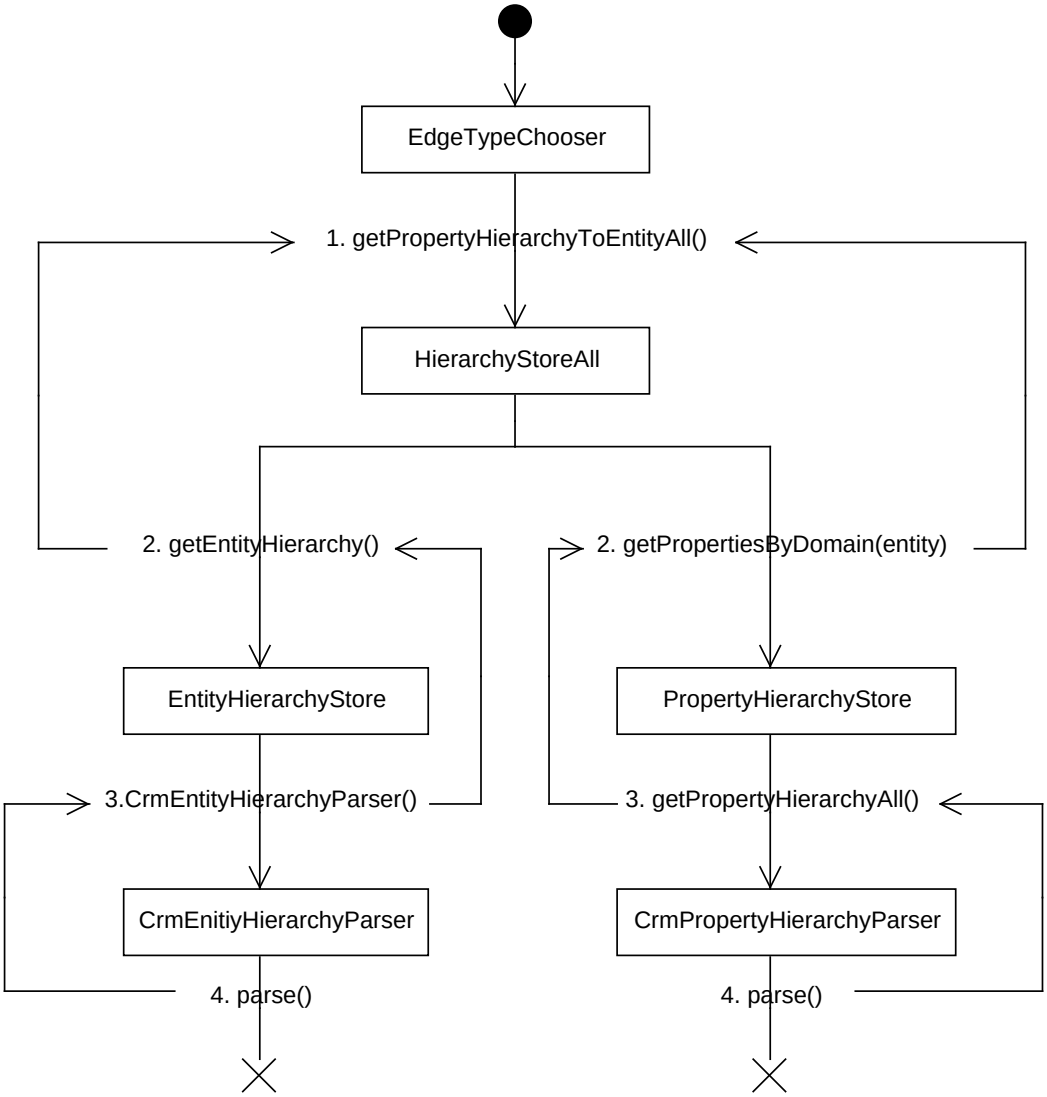

Abbildung 5.4.: Bestimmen der Zuordnung von Properties zu Entities

`ta` stellt einen Serializer-Aufruf dar, sie speichert die Parameter aus dem aufrufenden Transformations-Template (wie in Kapitel 5.3 beschrieben) sowie die zugewiesenen Entities (Domain und Range, siehe Kapitel 3.4). Für die eigentliche Ausgabe ruft die Methode `saveAsXSLT()` jetzt die Methoden der Klasse `Outputter` auf: Die Methode `out_1` gibt den XML-Header und den Namespace-Bereich aus. Die Methode `out_2` dient zur Ausgabe der Parameter, die bei Anwendung des generierten Skripts aus einem übergeordneten Skript befüllt werden. Die Methode `out_3` wird für jede Kante des Graphen einmal aufgerufen und erzeugt den der

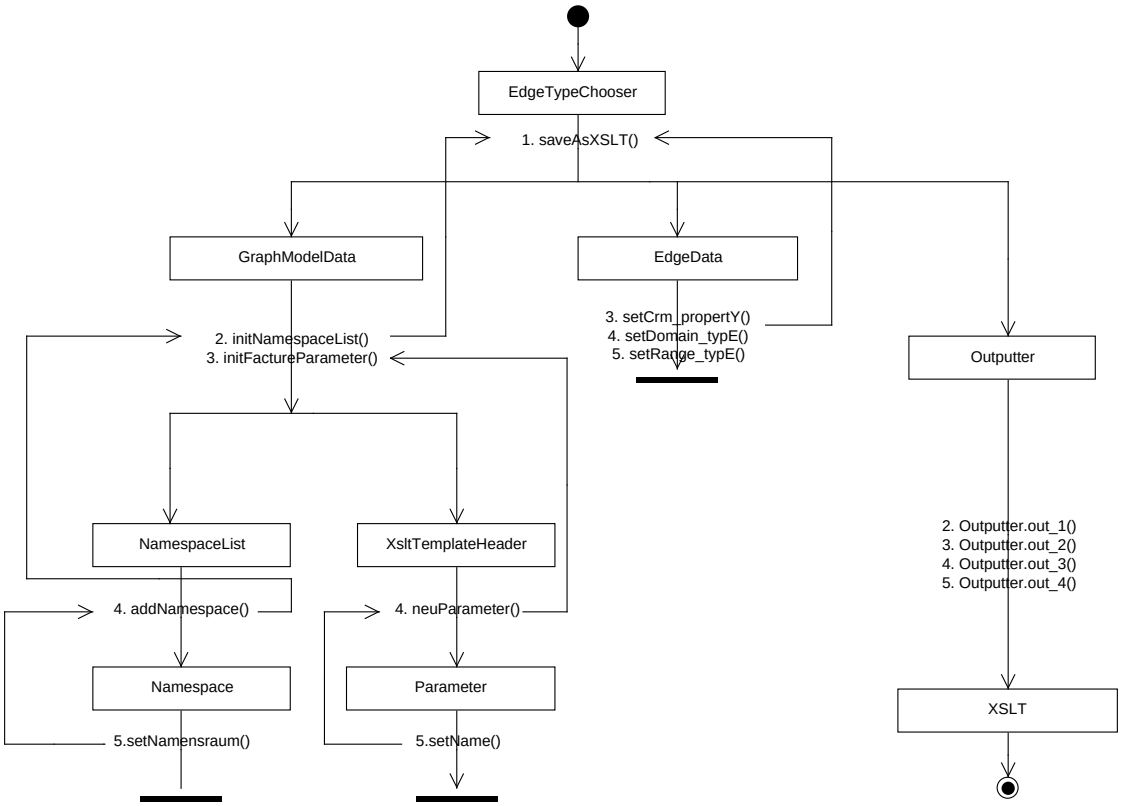

Abbildung 5.5.: Produktion des Transformation Template

Kante entsprechenden XSL-Aufruf des RDF-Serializers. Zum Schluss wird dann noch die Methode `Outputter.out_4()` aufgerufen, die den Schluss des XSLT-Skripts ausgibt.

5.4. User Interface

Im diesen Kapitel wird der Editor und seine grafische Benutzeroberfläche erläutert. Als Erstes soll ein Überblick über das Hauptfenster gegeben werden. Das Hauptfenster ist in drei Bereiche untergliedert. Links unten ist der Parameter-Bereich, links oben der Namespace-Bereich und auf der rechten Seite ist der Graph-Bereich, siehe Abbildung 5.6.

5.4.1. Namespace-Bereich

Der Namespace-Bereich gibt schon eine vordefinierte Menge an Namensräumen vor (siehe Kapitel 3.3), da aber möglicherweise noch anwendungsspezifische Änderungen vorzunehmen

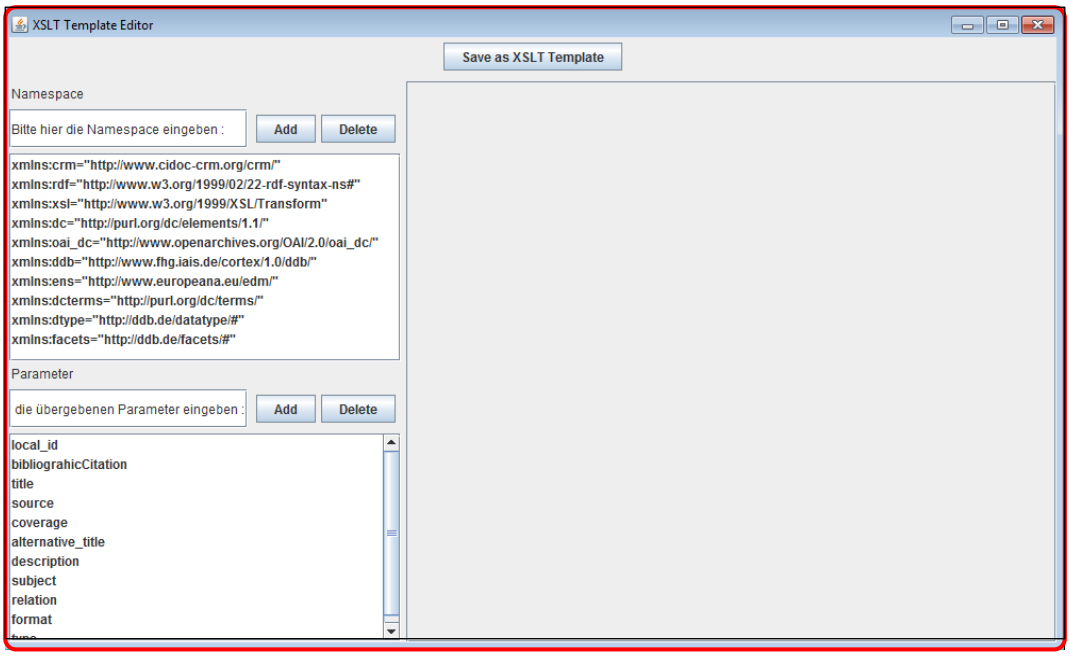

Abbildung 5.6.: Hauptfenster

sind, gibt es hier die Möglichkeit, diese zu editieren (siehe Abbildung 5.4.1). Sie werden dann auch im Template aktualisiert (siehe Kapitel 5.3).

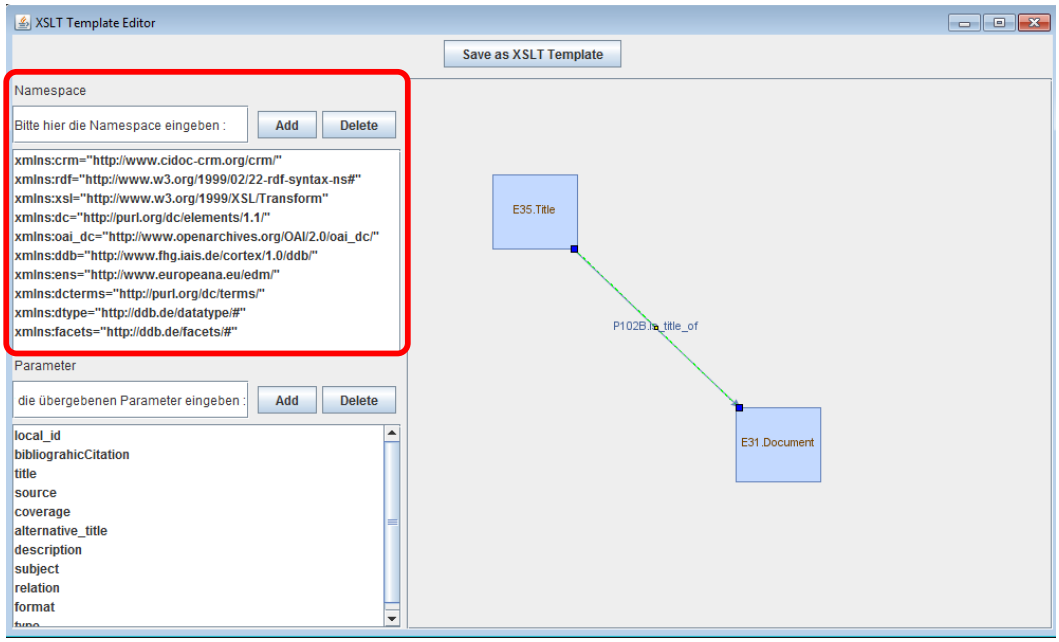

Abbildung 5.7.: Namespace

5.4.2. Parameter-Bereich

Die Liste der Parameter ergibt sich aus dem aufrufenden Template und kann in diesem Bereich eingegeben werden (siehe Abbildung 5.9). Diese Parameter können dann während der Konstruktion des Graphen anhand der Ontologie übergeben werden.

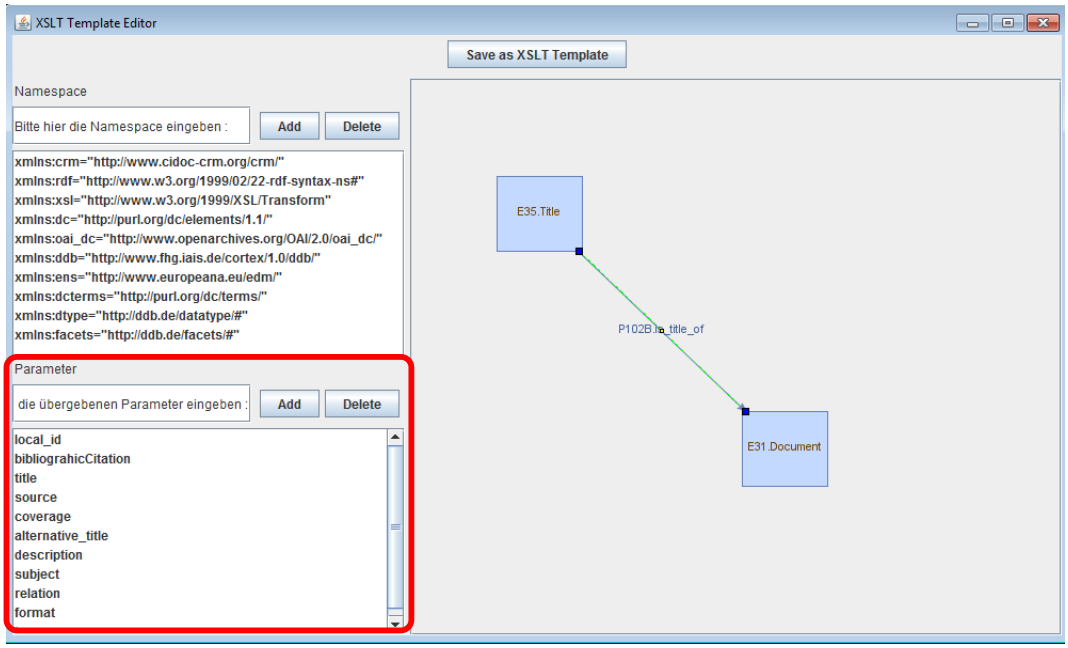

Abbildung 5.8.: Parameter-Bereich

5.4.3. Graph-Bereich

Im diesen Bereich kann ein Graph erzeugt werden, welcher der CIDOC-CRM Ontologie entspricht.

5.4.4. Knoten

Zum Setzen eines Knotens klickt man mit der linken Maustaste auf eine freie Stelle des Graph-Bereichs. Daraufhin öffnet sich der Dialog für die Entity-Eigenschaften, hier ist anhand der Ontologie der zuvor gesetzte Parameter und ein Entity-Typ zu wählen. Die zur Auswahl stehenden Entity-Typen werden wie im Kapitel 5.3.1 bestimmt.

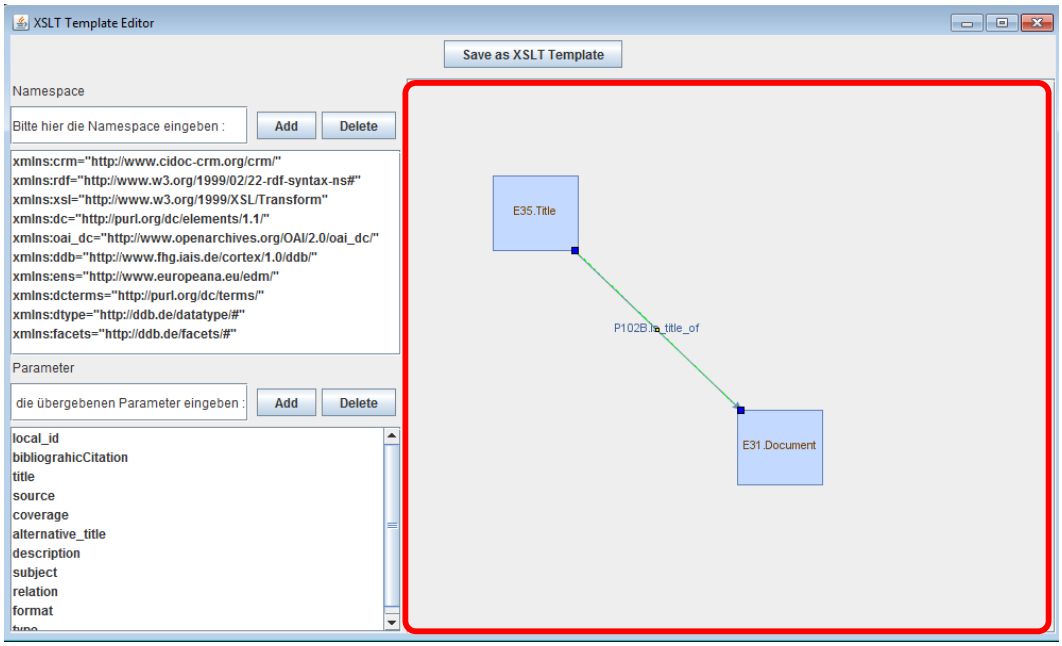

Abbildung 5.9.: Graph Bereich

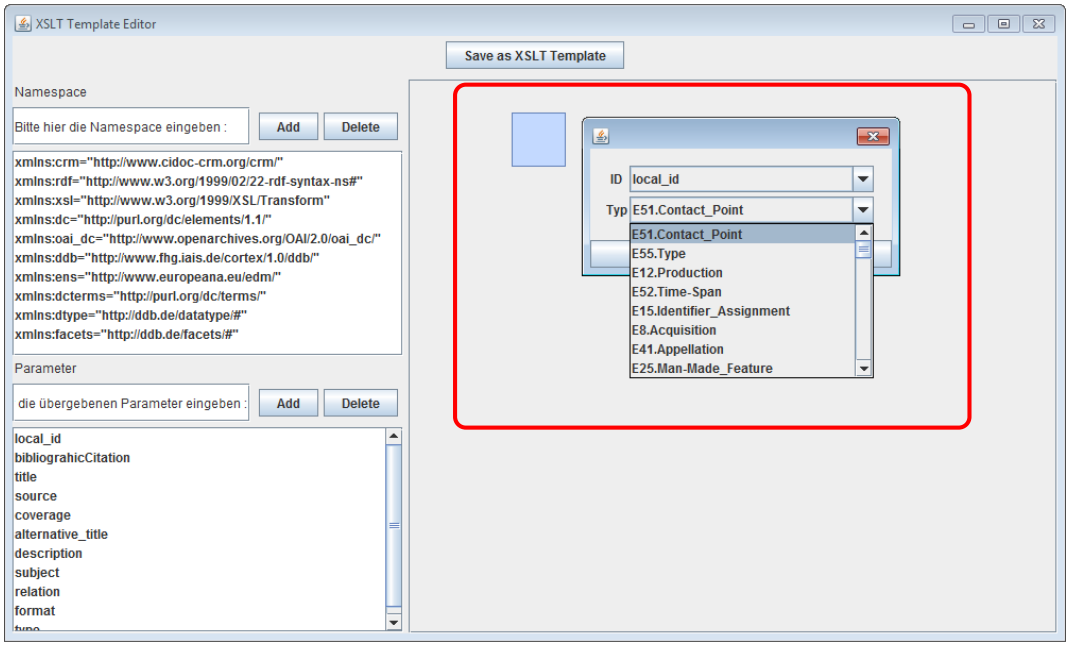

Abbildung 5.10.: Dialog für die Entity-Eigenschaften

5.4.5. Kanten

Zwischen je zwei Knoten kann eine Kante gesetzt werden. Wenn die Maus über den Knoten geht, der aus Sicht der Ontologie die Domain darstellt, siehe Kapitel 3.4, verändert sich der Mauszeiger zur einer Hand. Die linke Maustaste wird gedrückt und zu dem Knoten gezogen, den man mit dem ersten verbinden möchte. Beim Loslassen der Maustaste öffnet sich der Dialog zur Auswahl des Property-Typs (siehe Abbildung 5.11), die hier aufgeführten Property-Typen berechnen sich wie in Kapitel 5.3.3 beschrieben.

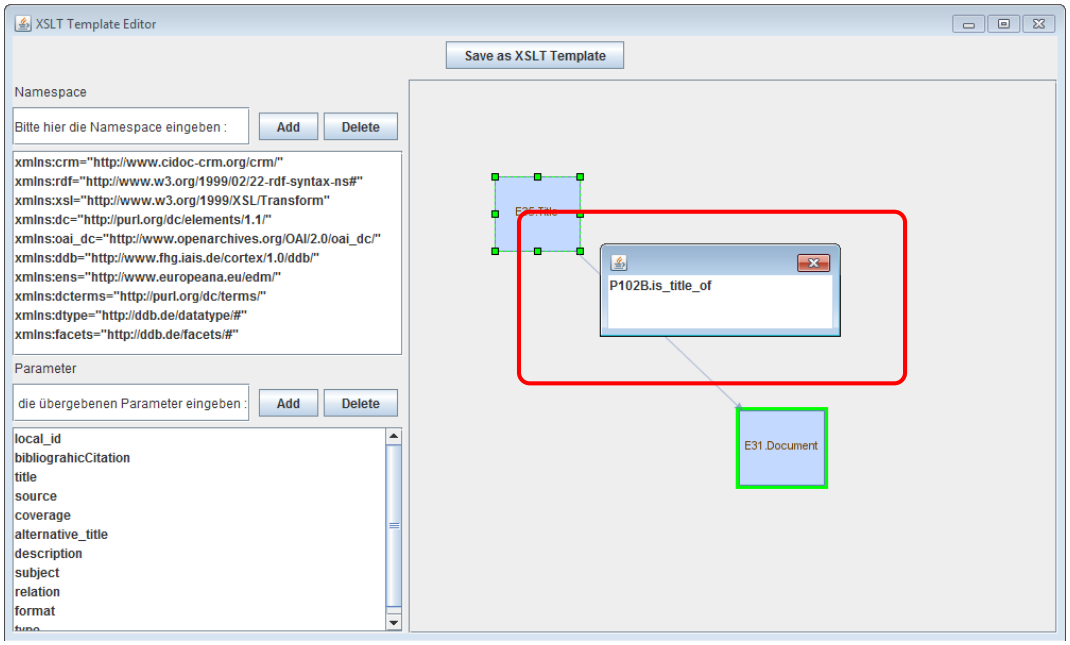

Abbildung 5.11.: Auswahl des Property-Typs

5.4.6. Kanten-Parameter

Mit einem Rechtsklick auf eine Kante öffnet sich ein weiterer Dialog, in dem die Parameter für den zugehörigen Aufruf des RDF-Serializers gesetzt werden können.

5.4.7. Transformations-Template erstellen

Zum Erstellen des Transformations-Templates kann nach Abschluss der Graph-Erstellung der Speichern-Knopf (siehe Abbildung 5.13) betätigt werden. Daraufhin kann noch der Pfad zum Speicherort angegeben werden.

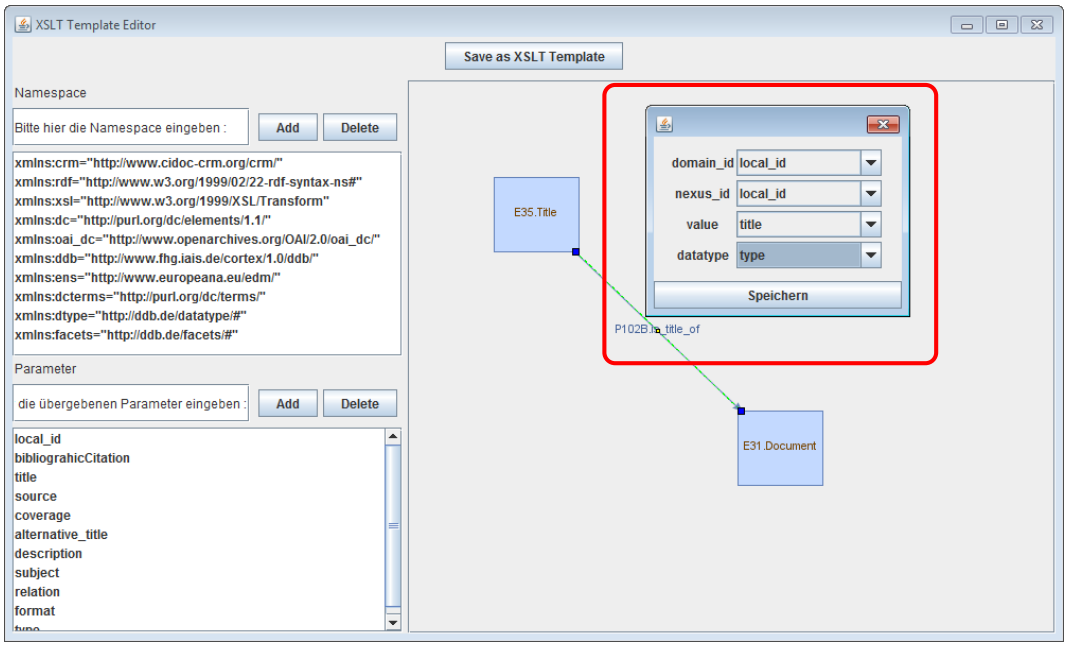

Abbildung 5.12.: Kanten-Parameter

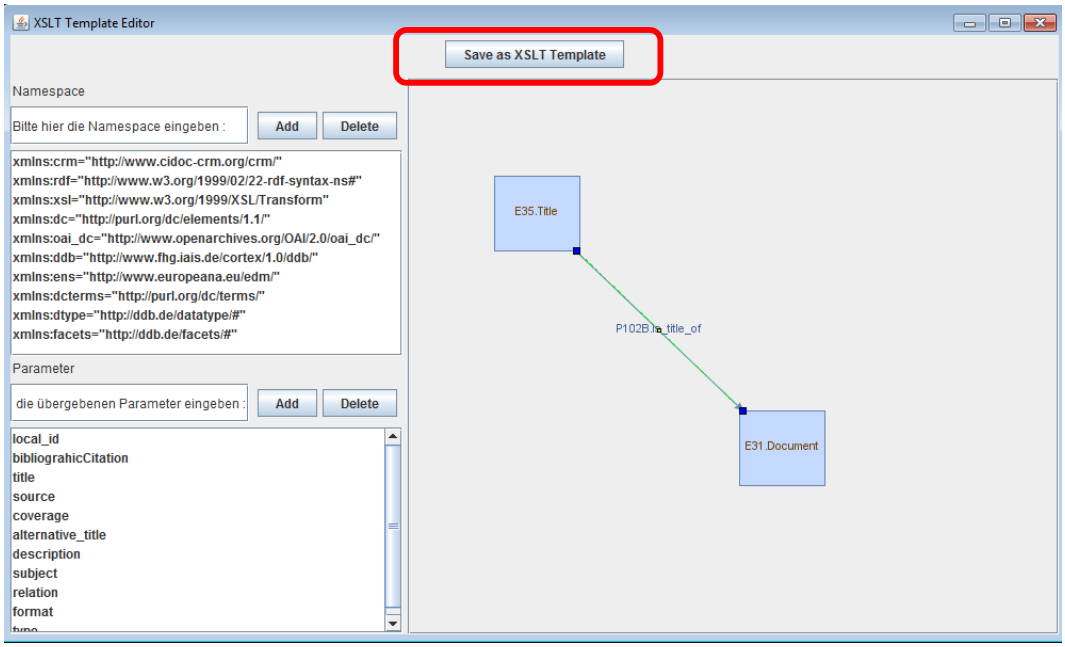

Abbildung 5.13.: Speichern-Knopf

Listing 5.6: Ergebnis Transformation-Template

```
<!--XSLT Template-->

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsl:stylesheet xmlns:crm="http://www.cidoc-crm.org/crm/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="2.0">

  <xsl:template name="XSLT Template">
    <xsl:with-param name="local_id"/>
    <xsl:with-param name="title"/>
    <xsl:with-param name="type"/>

    <xsl:call-template name="rdf_serializer">
      <xsl:with-param name="crm_property" select=
          "'crm:P102B.is_title_of'"/>
      <xsl:with-param name="range_type" select=
          "'http://www.cidoc-crm.org/crm-concepts/#E31'"/>
      <xsl:with-param name="value" select="\$title"/>
      <xsl:with-param name="domain_id" select="\$local_id"/>
      <xsl:with-param name="domain_type" select=
          "'http://www.cidoc-crm.org/crm-concepts/#E35'"/>
      <xsl:with-param name="datatype" select="\$type"/>
      <xsl:with-param name="nexus_id" select="\$local_id"/>
    </xsl:call-template>
  </xsl:template>
</xsl:stylesheet>
```

6. Evaluation

6.1. Softwaretest

6.1.1. Unit- und Integrationstests

Für das Paket XHierarchie gibt es drei Unit-Test die mit JUnit erstellt worden. Diese prüfen, ob es zur einer bestimmten Anfrage, wie zum Beispiel der Unterklassen einer Entity, mit der zu erwarteten Lösung, siehe Anhang A.1, übereinstimmt.

Test „EntityHierarchyStore“:

1. „testGetAllSuperClasses“

   Anfage „http://www.cidoc-crm.org/crm-concepts/#E12.Production“

   Ergebnis

   „http://www.cidoc-crm.org/crm-concepts/#E4.Period,
   http://www.cidoc-crm.org/crm-concepts/#E5.Event,
   http://www.cidoc-crm.org/crm-concepts/#E12.Production,
   http://www.cidoc-crm.org/crm-concepts/#E1.CRM_Entity,
   http://www.cidoc-crm.org/crm-concepts/#E2.Temporal_Entity,
   http://www.cidoc-crm.org/crm-concepts/#E7.Activity,
   http://www.cidoc-crm.org/crm-concepts/#E63.Beginning_of_Existence,
   http://www.cidoc-crm.org/crm-concepts/#E11.Modification.“

2. „testGetEntityHierarchy“

   Anfrage „http://www.cidoc-crm.org/crm-concepts/#E12.Production“

   Ergebnis

   „http://www.cidoc-crm.org/crm-concepts/#E4.Period,
   http://www.cidoc-crm.org/crm-concepts/#E5.Event,
   http://www.cidoc-crm.org/crm-concepts/#E12.Production,
   http://www.cidoc-crm.org/crm-concepts/#E1.CRM_Entity,

http://www.cidoc-crm.org/crm-concepts/#E2.Temporal_Entity,
http://www.cidoc-crm.org/crm-concepts/#E7.Activity,
http://www.cidoc-crm.org/crm-concepts/#E63.Beginning_of_Existence,
http://www.cidoc-crm.org/crm-concepts/#E11.Modification“

Test „HierarchyStoreTest“ 1: Der Test erstellt ein Set und weist ihm den Inhalt der Anfrage zu, dann prüft er, ob die zur Anfrage zu erwarteten Properties im Set sind.

1. **Anfrage** „http://www.cidoc-crm.org/crm-concepts/#E1.CRM_Entity", true“

   Ergebnis

   http://www.cidoc-crm.org/crm-concepts/#P41B.was_classified_by
   http://www.cidoc-crm.org/crm-concepts/#P2F.has_type
   http://www.cidoc-crm.org/crm-concepts/#P15B.influenced
   http://www.cidoc-crm.org/crm-concepts/#P138B.has_representation
   http://www.cidoc-crm.org/crm-concepts/#P67B.is_referred_to_by
   http://www.cidoc-crm.org/crm-concepts/#P129B.is_subject_of
   http://www.cidoc-crm.org/crm-concepts/#P141B.was_assigned_by
   http://www.cidoc-crm.org/crm-concepts/#P136B.supported_type_creation
   http://www.cidoc-crm.org/crm-concepts/#P137F.exemplifies
   http://www.cidoc-crm.org/crm-concepts/#P3F.has_note
   http://www.cidoc-crm.org/crm-concepts/#P62B.is_depicted_by
   http://www.cidoc-crm.org/crm-concepts/#P17B.motivated"
   http://www.cidoc-crm.org/crm-concepts/#P70B.is_documented_in
   http://www.cidoc-crm.org/crm-concepts/#P39B.was_measured_by
   http://www.cidoc-crm.org/crm-concepts/#P1F.is_identified_by
   http://www.cidoc-crm.org/crm-concepts/#P48F.has_preferred_identifier
   http://www.cidoc-crm.org/crm-concepts/#P140B.was_attributed_by

Test „HierarchyStoreTest“ 2: Dieser Test prüft, ob zur einer bestimmten Domain und Range die dazugehörigen Properties wiedergegeben werden.

1. „testgetPredicatesByDomainAndRange“

   Anfrage

   „http://www.cidoc-crm.org/crm-concepts/#E12.Production,“,
   „http://www.cidoc-crm.org/crm-concepts/#E1.CRM_Entity“

Ergebnis

http://www.cidoc-crm.org/crm-concepts/#P15F.was_influenced_by
http://www.cidoc-crm.org/crm-concepts/#P17F.was_motivated_by

Test „PropertyHierarchyStoreTest“ Bei diesen Test wird überprüft, ob zu einer bestimmten Property die richtigen Superklassen sowie ihre Hierarchie ermittelt wird.

1. „testGetSuperProperties“

   Anfrage „http://www.cidoc-crm.org/crm-concepts/#P22F.transferred_title_to“

   Ergebnis

   http://www.cidoc-crm.org/crm-concepts/#P14F.carried_out_by
   http://www.cidoc-crm.org/crm-concepts/#P12F.occurred_in_the_presence_of
   http://www.cidoc-crm.org/crm-concepts/#P11F.had_participant

1. „testGetPropertyHierarchy“

   Anfrage „http://www.cidoc-crm.org/crm-concepts/#P22F.transferred_title_to“

   Ergebnis

   http://www.cidoc-crm.org/crm-concepts/#P22F.transferred_title_to
   http://www.cidoc-crm.org/crm-concepts/#P14F.carried_out_by
   http://www.cidoc-crm.org/crm-concepts/#P12F.occurred_in_the_presence_of
   http://www.cidoc-crm.org/crm-concepts/#P12F.occurred_in_the_presence_of

7. Fazit

Ziel der Arbeit war es herauszufinden, ob es möglich ist, Transformations-Templates zur Produktion semantischer Relationen mit einem grafischen Editor zu produzieren. Nach Festlegung der Vorgehensweise wurde in einzelnen Schritten gezeigt, dass eine Umsetzung in einen Graphen ein möglicher Weg ist. Wie sich zeigte, lassen sich Entities und Properties der im CIDOC CRM beschriebenen Ontologie als Knoten und Kanten darstellen und ihre richtige Verwendung prüfen.

Bei der Auswahl des Framework zur Repräsentation des Graphen zeigte sich, dass die zur Verfügung stehende Zeit eine große Rolle spielt. Bei der Umsetzung der Logik der Hierarchien gab es einen Wechsel zu einer performanteren Lösung. Die Umsetzung von der grafischen in die textuelle Darstellung konnte durch Abfrage des Graphen sowie Verwendung von zusätzlichen Daten (Liste der Namespaces und Liste der Parameter) realisiert werden. Das MVC-Muster konnte nicht im Ganzen umgesetzt werden, aber die grundsätzlich sinnvolle Trennung der grafischen von der logischen Schicht wurde erreicht. Damit ist diese Arbeit ein Erfolg, denn sie hat gezeigt, dass es möglich ist, einen grafischen Editor zur Unterstützung des Metadaten-Mappings zu entwickeln.

Der im Rahmen dieser Arbeit entwickelte Editor ist als Baustein einer Gesamtlösung anzusehen, die letztlich die 30.000 Kultureinrichtungen in Deutschland in die Lage versetzt, notwendige Änderungen an den Transformation Skripten eigenständig vorzunehmen.

8. Ausblick

Durch die geringen Anforderungen an die Repräsentation der Ontologie könnte der im Rahmen dieser Arbeit entwickelte Editor auch für andere Ontologien als CIDOC CRM Anwendung finden. Eine aus Anwendersicht heraus nützliche Erweiterung ist es, die Auswahl von Properties in Hinblick auf ihre Range zu überprüfen, um weitere Klassen von Eingabefehlern auszuschließen.

Wie schon im Kapitel 1.4 beschrieben, ist im DDB-Projekt der nächste Schritt den Prototypen in das Projekt zu integrieren. Hierzu gibt es zwei Alternativen: Entweder muss die Oberfläche durch eine neue Eclipse-basierte Oberfläche ersetzt werden oder es muss versucht werden, die jetzige Swing-basierte Oberfläche in die Eclipse-eigene SWT-Oberfläche zu integrieren. Um eine Gesamtlösung im Sinne von Kapitel 7 realisieren zu können, müssen außerdem noch Generatoren für die darüberliegenden Schichten der Transformations-Skripte entwickelt werden.

Literatur

[Amm02] Dirk Ammelburger. *XML mit Java*. München Germany: Markt und Technik Verlag, 2002. ISBN: 9783827263278.

[BL+05] Tim Berners-Lee u. a. *Spinning the semantic web: Bringing the World Wide Web to its full potential*. Cambridge London: MIT Press, 2005. ISBN: 9780262562126.

[BP08] Helmut Balzert und Jürgen Priemer. *Java 6 Anwendungen programmieren: Von der GUI-Programmierung bis zur Datenbank-Anbindung*. Informatik. Herdecke u.a: W3L Verlag, 2008. ISBN: 9783937137094.

[Bon08] Frank Bongers. *XSLT 2.0 & XPath 2.0*. 2 Auflage. Galileo computing. Bonn: Galileo Press, 2008. ISBN: 3898426947.

[CID11] Heinz Lampe Karl, Siegfried Krause und Martin Doerr. *CIDOC Conceptual Reference Model*. Hrsg. von ICOM Deutschland Beiträge zur Museologie Band 1. 2011. URL: `http://www.icom-deutschland.de/client/media/380/cidoccrm_end.pdf` (besucht am 14. 10. 2011).

[DDB10] Deutsche Digitale Bibliothek. *Rahmenbedingungen zur Anforderungsanalyse aus politischer, rechtlicher und funktionaler/technischer Sicht*. 2010. URL: `http://www.deutsche-digitale-bibliothek.de/pdf/rahmenbedingungen_zur_anforderungsanalyse_aus_politischer_rechtlicher_funktionaler_technischer_sicht_finale_fassung_2010.05.21.pdf` (besucht am 14. 10. 2011).

[DNB11] Deutsche Nationalbibliothek. *Partner. Leibniz-Institut für Informatik FIZ*. 2011. URL: `http://www.deutsche-digitale-bibliothek.de/partner.htm` (besucht am 14. 10. 2011).

[DOM11] Document Object Model. *DOM*. 2011. URL: `http://www.w3.org/DOM/` (besucht am 14. 10. 2011).

[EE04] Rainer Eckstein und Silke Eckstein. *XML und Datenmodellierung: XML-Schema und RDF zur Modellierung von Daten und Metadaten einsetzen*. 1. Aufl. xml.bibliothek. Heidelberg: dpunkt Verlag, 2004. ISBN: 3898642224.

[FG11] Fraunhofer-Gesellschaft. *Über Fraunhofer*. 2011. URL: http://www.fraunhofer.de/ueber-fraunhofer/ (besucht am 14. 10. 2011).

[FH08] Peter Fischer und Peter Hofer. *Lexikon der Informatik*. 14. Berlin Heidelberg: Springer-Verlag, 2008. ISBN: 3540725504.

[GP00] Charles F. Goldfarb und Paul Prescod. *Das XML-Handbuch*. 2. Auflage. München und Boston: Addison Wesley, 2000. ISBN: 9783827317124.

[JDO11] JDOM. 2011. URL: http://www.jdom.org/ (besucht am 14. 10. 2011).

[Jen11] Jena. *Ein Semantic Web Framework*. 2011. URL: http://jena.sourceforge.net/ (besucht am 14. 10. 2011).

[Min09a] Ministerpräsidentenkonferenz. *Gemeinsame Eckpunkte Bund, Länder und Kommunen 2. Dezember 2009*. 2009. (Besucht am 14. 10. 2011).

[Min09b] Ministerpräsidentenkonferenz. *Verwaltungs- und Finanzabkommen: gemäß Beschluss der Ministerpräsidentenkonferenz vom 26.03.2009*. 2009. URL: http://www.deutsche-digitale-bibliothek.de/pdf/verwaltungs_und_finanzabkommen_finale%20Fassung02122009.pdf (besucht am 14. 10. 2011).

[Pel06] Tassilo Pellegrini, Hrsg. *Semantic Web: Wege zur vernetzten Wissensgesellschaft ; mit 4 Tabellen*. Berlin, Heidelberg, New York: Springer, 2006. ISBN: 3540293248.

[RDF04] World Wide Web Consortium. *Resource Description Framework (RDF)*. 2004. URL: http://www.w3.org/RDF/ (besucht am 14. 10. 2011).

[SB10] Kai Stalmann und Reinhard Budde. *Grobkonzept für das Portal der Deutschen Digitalen Bibliothek*. 2010. URL: http://www.iais.fraunhofer.de/uploads/media/DDBGrobkonzeptFinal.pdf (besucht am 14. 10. 2011).

[Su11] Kai Stalmann und u.a. *"Kultur Wissennetz"Die Architektur der Deutschen Digitalen Bibliothek. In: EMISA-Forum*. 2011. URL: http://publica.fraunhofer.de/documents/N-174647.html (besucht am 14. 10. 2011).

[Ull04] Mike Ullrich. „Taxonomie, Thesaurus,Topic Map, Ontologie - ein Vergleich“. 2004.

[CID11] CIDOC CRM. *Mappings, Specializations and Data Examples*. 2011. URL: http://www.cidoc-crm.org/crm_mappings.html (besucht am 14. 10. 2011).

[Fra11] Fraunhofer IAIS. *Über uns*. 2011. URL: http://www.iais.fraunhofer.de/profil.html (besucht am 14. 10. 2011).

A. Anhang

A.1. CIDOC-CRM Entities und Properties

CIDOC CRM Class Hierarchy
E1 CRM Entity
E2 - Temporal Entity
E3 - - Condition State
E4 - - Period
E5 - - - Event
E7 - - - - Activity
E8 - - - - - Acquisition Event
E9 - - - - - Move
E10 - - - - - Transfer of Custody
E11 - - - - - Modification
E12 - - - - - - Production
E79 - - - - - - Part Addition
E80 - - - - - - Part Removal
E13 - - - - - Attribute Assignment
E14 - - - - - - Condition Assessment
E15 - - - - - - Identifier Assignment
E16 - - - - - - Measurement
E17 - - - - - - Type Assignment
E65 - - - - - Creation
E83 - - - - - - Type Creation
E66 - - - - - Formation
E85 - - - - - Joining
E86 - - - - - Leaving
E87 - - - - - Curation Activity
E63 - - - - Beginning of Existence
E67 - - - - - Birth
E81 - - - - - Transformation
E12 - - - - - Production
E65 - - - - - Creation
E83 - - - - - - Type Creation
E66 - - - - - Formation
E64 - - - - End of Existence
E6 - - - - - Destruction
E68 - - - - - Dissolution
E69 - - - - - Death
E81 - - - - - Transformation
E77 - Persistent Item
E70 - - Thing
E72 - - - Legal Object
E18 - - - - Physical Thing
E19 - - - - - Physical Object
E20 - - - - - - Biological Object
E21 - - - - - - - Person
E22 - - - - - - Man-Made Object
E84 - - - - - - - Information Carrier
E24 - - - - - Physical Man-Made Thing
E22 - - - - - - Man-Made Object
E84 - - - - - - - Information Carrier
E25 - - - - - - Man-Made Feature
E78 - - - - - - Collection
E26 - - - - - Physical Feature
E27 - - - - - - Site
E25 - - - - - - Man-Made Feature
E90 - - - - Symbolic Object
E73 - - - - - Information Object
E29 - - - - - Design or Procedure

Abbildung A.1.: Entities Seite 1 angelehnt an [Abb. CID11]

E31 - - - - - Document
Definition of the CIDOC Conceptual Reference Model xxiii
E32 - - - - - - - Authority Document
E33 - - - - - - Linguistic Object
E34 - - - - - - - Inscription
E35 - - - - - - - Title
E36 - - - - - Visual Item
E37 - - - - - - - Mark
E34 - - - - - - - - Inscription
E38 - - - - - - - Image
E41 - - - - - Appellation
E42 - - - - - - Identifier
E44 - - - - - - Place Appellation
E45 - - - - - - - Address
E46 - - - - - - - Section Definition
E47 - - - - - - - Spatial Coordinates
E48 - - - - - - - Place Name
E49 - - - - - - Time Appellation
E50 - - - - - - - Date
E75 - - - - - - Conceptual Object Appellation
E82 - - - - - - Actor Appellation
E51 Contact Point
E45 Address
E35 - - - - Title
E71 - - - Man-Made Thing
E24 - - - - Physical Man-Made Thing
E22 - - - - - Man-Made Object
E84 - - - - - - Information Carrier
E25 - - - - - Man-Made Feature
E78 - - - - - Collection
E28 - - - - Conceptual Object
E90 - - - - - Symbolic Object
E73 - - - - - - Information Object
E29 - - - - - - - Design or Procedure
E31 - - - - - - - Document
E32 - - - - - - - - Authority Document
E33 - - - - - - Linguistic Object
E34 - - - - - - - - Inscription
E35 - - - - - - - Title
E36 - - - - - - - - Visual Item
E37 - - - - - - - - Mark
E34 - - - - - - - - - Inscription
E38 - - - - - - - - Image
E41 - - - - - - Appellation
E42 - - - - - - - Identifier
E44 - - - - - - - Place Appellation
E45 - - - - - - - - Address
E46 - - - - - - - - Section Definition
E47 - - - - - - - - Spatial Coordinates
E48 - - - - - - - - Place Name
E49 - - - - - - Time Appellation
E50 - - - - - - - - Date
E75 - - - - - - - Conceptual Object Appellation
E82 - - - - - - - Actor Appellation
E51 - - - - - - - Contact Point
E45 - - - - - - - - Address
E35 - - - - - Title

Abbildung A.2.: Entities Seite 2 angelehnt an [Abb. CID11]

E89 Propositional Object
E73 - - - - - - *Information Object*
E29 - - - - - - - *Design or Procedure*
E31 - - - - - - - *Document*
E32 - - - - - - - - *Authority Document*
Definition of the CIDOC Conceptual Reference Model xxiv
E33 - - - - - - - *Linguistic Object*
E34 - - - - - - - - *Inscription*
E35 - - - - - - - - *Title*
E36 - - - - - - - *Visual Item*
E37 - - - - - - - - *Mark*
E34 - - - - - - - - *Inscription*
E38 - - - - - - - - *Image*
E30 - - - - - - Right
E55 - - - - - Type
E56 - - - - - - Language
E57 - - - - - - Material
E58 - - - - - - Measurement Unit
E39 - - Actor
E74 - - - Group
E40 - - - - Legal Body
E21 - - - *Person*
E52 - Time-Span
E53 - Place
E54 - Dimension
E59 Primitive Value
E60 - Number
E61 - Time Primitive
E62 - String

Abbildung A.3.: Entities Seite 3 angelehnt an [Abb. CID11]

CIDOC CRM Property Hierarchy:

Property id	Property Name	Entity – Domain	Entity - Range
P1	is identified by (identifies)	E1 CRM Entity	E41 Appellation
P48	- has preferred identifier (is preferred identifier of)	E1 CRM Entity	E42 Identifier
P78	- is identified by (identifies)	E52 Time-Span	E49 Time Appellation
P87	- is identified by (identifies)	E53 Place	E44 Place Appellation
P102	-has title (is title of)	E71 Man-Made Thing	E35 Title
P131	-is identified by (identifies)	E39 Actor	E82 Actor Appellation
P2	has type (is type of)	E1 CRM Entity	E55 Type
P137	-exemplifies (is exemplified by)	E1 CRM Entity	E55 Type
P3	has note	E1 CRM Entity	E62 String
P79	- beginning is qualified by	E52 Time-Span	E62 String
P80	- end is qualified by	E52 Time-Span	E62 String
P4	has time-span (is time-span of)	E2 Temporal Entity	E52 Time-Span
P5	consists of (forms part of)	E3 Condition State	E3 Condition State
P7	took place at (witnessed)	E4 Period	E53 Place
P26	- moved to (was destination of)	E9 Move	E53 Place
P27	- moved from (was origin of)	E9 Move	E53 Place
P8	took place on or within (witnessed)	E4 Period	E19 Physical Object
P9	consists of (forms part of)	E4 Period	E4 Period
P10	falls within (contains)	E4 Period	E4 Period
P12	occurred in the presence of (was present at)	E5 Event	E77 Persistent Item
P11	- had participant (participated in)	E5 Event	E39 Actor
P14	- - carried out by (performed)	E7 Activity	E39 Actor
P22	- - - transferred title to (acquired title through)	E8 Acquisition	E39 Actor
P23	- - - transferred title from (surrendered title through)	E8 Acquisition	E39 Actor
P28	- - - custody surrendered by (surrendered custody through)	E10 Transfer of Custody	E39 Actor
P29	- - - custody received by (received custody through)	E10 Transfer of Custody	E39 Actor
P96	- - by mother (gave birth)	E67 Birth	E21 Person
P99	- - dissolved (was dissolved by)	E68 Dissolution	E74 Group
P143	- - joined (was joined by)	E85 Joining	E39 Actor
P144	- - joined with (gained member by)	E85 Joining	E74 Group
P145	- - separated (left by)	E86 Leaving	E39 Actor
P146	- - separated from (lost member by)	E86 Leaving	E74 Group
P16	- used specific object (was used for)	E7 Activity	E70 Thing
P33	- - used specific technique (was used by)	E7 Activity	E29 Design or Procedure
P142	- - used constituent (was used in)	E15 Identifier Assignment	E41 Appellation
P25	- moved (moved by)	E9 Move	E19 Physical Object
P31	- has modified (was modified by)	E11 Modification	E24 Physical Man-Made Thing
P108	- - has produced (was produced by)	E12 Production	E24 Physical Man-Made Thing
P110	- - augmented (was augmented by)	E79 Part Addition	E24 Physical Man-Made Thing
P112	- - diminished (was diminished by)	E80 Part Removal	E24 Physical Man-Made Thing
P92	- brought into existence (was brought into existence by)	E63 Beginning of Existence	E77 Persistent Item
P94	- - has created (was created by)	E65 Creation	E28 Conceptual Object
P135	- - - created type (was created by)	E83 Type Creation	E55 Type
P95	- - has formed (was formed by)	E66 Formation	E74 Group
P98	- - brought into life (was born)	E67 Birth	E21 Person
P108	- - *has produced (was produced by)*	*E12 Production*	*E24 Physical Man-Made Thing*
P123	- - resulted in (resulted from)	E81 Transformation	E77 Persistent Item
P93	- took out of existence (was taken out of existence by)	E64 End of Existence	E77 Persistent Item
P13	- - destroyed (was destroyed by)	E6 Destruction	E18 Physical Thing
P99	- - *dissolved (was dissolved by)*	*E68 Dissolution*	*E74 Group*
P100	- - was death of (died in)	E69 Death	E21 Person
P124	- - transformed (was transformed by)	E81 Transformation	E77 Persistent Item
P142	- - *used constituent (was used in)*	*E15 Identifier Assignment*	*E41 Appellation*
P15	was influenced by (influenced)	*E7 Activity*	E1 CRM Entity
P16	- *used specific object (was used for)*	*E7 Activity*	*E70 Thing*
P33	- - *used specific technique (was used by)*	*E11 Modification*	*E29 Design or Procedure*
P142	- - *used constituent (was used in)*	*E15 Identifier Assignment*	*E41 Appellation*
P17	- was motivated by (motivated)	E7 Activity	E1 CRM Entity
P134	-continued (was continued by)	E7 Activity	E7 Activity
P136	-was based on (supported type creation)	E83 Type Creation	E1 CRM Entity
P19	was intended use of (was made for)	E7 Activity	E71 Man-Made Thing
P20	had specific purpose (was purpose of)	E7 Activity	E5 Event
P21	had general purpose (was purpose of)	E7 Activity	E55 Type
P24	transferred title of (changed ownership through)	E8 Acquisition	E18 Physical Thing
P30	transferred custody of (custody transferred through)	E10 Transfer of Custody	E18 Physical Thing
P43	has dimension (is dimension of)	E70 Thing	E54 Dimension
P44	has condition (is condition of)	E18 Physical Thing	E3 Condition State
P45	consists of (is incorporated in)	E18 Physical Thing	E57 Material
P46	is composed of (forms part of)	E18 Physical Thing	E18 Physical Thing
P56	- bears feature (is found on)	E19 Physical Object	E26 Physical Feature
P49	has former or current keeper (is former or current keeper of)	E18 Physical Thing	E39 Actor

Abbildung A.4.: Properties to Entities Seite 1 angelehnt an [Abb. CID11]

Property id	Property Name	Entity – Domain	Entity - Range
P50	- has current keeper (is current keeper of)	E18 Physical Thing	E39 Actor
P51	has former or current owner (is former or current owner of)	E18 Physical Thing	E39 Actor
P52	- has current owner (is current owner of)	E18 Physical Thing	E39 Actor
P53	has former or current location (is former or current location of)	E18 Physical Thing	E53 Place
P55	-has current location (currently holds)	E19 Physical Object	E53 Place
P54	has current permanent location (is current permanent location of)	E19 Physical Object	E53 Place
P57	has number of parts	E19 Physical Object	E60 Number
P58	has section definition (defines section)	E18 Physical Thing	E46 Section Definition
P59	has section (is located on or within)	E18 Physical Thing	E53 Place
P62	depicts (is depicted by)	E24 Physical Man-Made Thing	E1 CRM Entity
P67	refers to (is referred to by)	E89 Propositional Object	E1 CRM Entity
P70	-documents (is documented in)	E31 Document	E1 CRM Entity
P71	-lists (is listed in)	E32 Authority Document	E55 Type
P129	-is about (is subject of)	E89 Propositional Object	E1 CRM Entity
P138	-represents (has representation)	E36 Visual Item	E1 CRM Entity
P68	foresees use of (use foreseen by)	E29 Design or Procedure	E57 Material
P69	is associated with	E29 Design or Procedure	E29 Design or Procedure
P72	has language (is language of)	E33 Linguistic Object	E56 Language
P74	has current or former residence (is current or former residence of)	E39 Actor	E53 Place
P75	possesses (is possessed by)	E39 Actor	E30 Right
P76	has contact point (provides access to)	E39 Actor	E51 Contact Point
P81	ongoing throughout	E52 Time-Span	E61 Time Primitive
P82	at some time within	E52 Time-Span	E61 Time Primitive
P83	had at least duration (was minimum duration of)	E52 Time-Span	E54 Dimension
P84	had at most duration (was maximum duration of)	E52 Time-Span	E54 Dimension
P86	falls within (contains)	E52 Time-Span	E52 Time-Span
P88	consists of (forms part of)	E53 Place	E53 Place
P89	falls within (contains)	E53 Place	E53 Place
P90	has value	E54 Dimension	E60 Number
P91	has unit (is unit of)	E54 Dimension	E58 Measurement Unit
P97	from father (was father for)	E67 Birth	E21 Person
P101	had as general use (was use of)	E70 Thing	E55 Type
P103	was intended for (was intention of)	E71 Man-Made Thing	E55 Type
P104	is subject to (applies to)	E72 Legal Object	E30 Right
P105	right held by (has right on)	E72 Legal Object	E39 Actor
P52	- has current owner (is current owner of)	E18 Physical Thing	E39 Actor
P106	is composed of (forms part of)	E90 Symbolic Object	E90 Symbolic Object
P107	has current or former member (is current or former member of)	E74 Group	E39 Actor
P109	has current or former curator (is current or former curator of)	E78 Collection	E39 Actor
P111	added (was added by)	E79 Part Addition	E18 Physical Thing
P113	removed (was removed by)	E80 Part Removal	E18 Physical Thing
P114	is equal in time to	E2 Temporal Entity	E2 Temporal Entity
P115	finishes (is finished by)	E2 Temporal Entity	E2 Temporal Entity
P116	starts (is started by)	E2 Temporal Entity	E2 Temporal Entity
P117	occurs during (includes)	E2 Temporal Entity	E2 Temporal Entity
P118	overlaps in time with (is overlapped in time by)	E2 Temporal Entity	E2 Temporal Entity
P119	meets in time with (is met in time by)	E2 Temporal Entity	E2 Temporal Entity
P120	occurs before (occurs after)	E2 Temporal Entity	E2 Temporal Entity
P121	overlaps with	E53 Place	E53 Place
P122	borders with	E53 Place	E53 Place
P125	used object of type (was type of object used in)	E7 Activity	E55 Type
P32	- used general technique (was technique of)	E7 Activity	E55 Type
P126	employed (was employed in)	E11 Modification	E57 Material
P127	has broader term (has narrower term)	E55 Type	E55 Type
P128	carries (is carried by)	E24 Physical Man-Made Thing	E73 Information Object
P65	-shows visual item (is shown by)	E24 Physical Man-Made Thing	E36 Visual Item
P130	shows features of (features are also found on)	E70 Thing	E70 Thing
P73	-has translation (is translation of)	E33 Linguistic Object	E33 Linguistic Object
P132	overlaps with	E4 Period	E4 Period
P133	is separated from	E4 Period	E4 Period
P139	has alternative form	E41 Appellation	E41 Appellation
P140	assigned attribute to (was attributed by)	E13 Attribute Assignment	E1 CRM Entity
P34	-concerned (was assessed by)	E14 Condition Assessment	E18 Physical Thing
P39	-measured (was measured by)	E16 Measurement	E1 CRM Entity
P41	-classified (was classified by)	E17 Type Assignement	E1 CRM Entity
P141	assigned (was assigned by)	E13 Attribute Assignement	E1 CRM Entity
P35	-has identified (identified by)	E14 Condition Assessment	E3 Condition State
P37	-assigned (was assigned by)	E15 Identifier Assignement	E42 Identifier
P38	-deassigned (was deassigned by)	E15 Identifier Assignement	E42 Identifier
P40	-observed dimension (was observed in)	E16 Measurement	E54 Dimension
P42	-assigned (was assigned by)	E17 Type Assignment	E55 Type
P147	curated (was curated by)	E87 Curation Activity	E78 Collection
P148	has component (is component of)	E89 Propositional Object	E89 Propositional Object

Abbildung A.5.: Properties to Entities Seite 2 angelehnt an [Abb. CID11]