

Projekt Deutsche Digitale Bibliothek (DDB)

Architekturdokumentation

2011-04-11 – v 1.4

Wir weisen darauf hin, dass dieses Dokument urheberrechtlich geschützte Inhalte und Know-How enthält, die Betriebsgeheimnisse des Fraunhofer-

Instituts IAIS darstellen. Das Dokument darf nur zur Information für das Cortex-System verwendet werden.

Jede Verwendung für andere Zwecke, Vervielfältigung, Modifikation und Weitergabe an Dritte ohne vorherige schriftliche Zustimmung von Fraunhofer IAIS verletzt die Rechte der Fraunhofer Gesellschaft und ist Gegenstand von Unterlassungs- und Schadensersatzansprüchen.

ZUSAMMENFASSUNG

Das Dokument stellt die Architektur der Plattform der Deutschen Digitalen Bibliothek dar.

Es beschreibt in Ergänzung zum Grobkonzept die Technologien, Komponenten, Schnittstellen, Prozesse, Frameworks und Datenmodelle, die zum Aufbau der Plattform verwendet werden, und skizziert die Infrastruktur zum Betrieb des geplanten Systems.

Das beschriebene System wurde entworfen, um eine große Zahl von Objekten des kulturellen Erbes aus den entsprechenden kulturellen und wissenschaftlichen Einrichtungen dieses Landes über das Internet zur Verfügung zu stellen. Die Objekte werden als Metadaten und ggf. binäre Derivate in heterogener Form geliefert und sollen in zeitgemäßer Form online recherchiert werden können.

Der Implementierung liegt ein einfaches Konzept zugrunde. Die heterogenen Eingangsmetadaten sollen in ein einheitliches Datenrepräsentationsmodell übersetzt werden, das eine Abbildung der Semantik der Originaldaten in möglichst verlustfreier Form erlaubt, aber von einzelnen Datenformaten und Metadatensprachen abstrahiert. Die Metadaten und Daten werden im Archiv der Plattform gespeichert. Das Modell wird für verschiedene Zwecke genutzt: z.B. um exportierbare Formate daraus zu erzeugen (etwa das Europeana Data Modell), um eine Deduplikation von Objekten anzustreben, um die Objekte zu vernetzen etc. Außerdem dient das Modell als Ausgangspunkt für ein weiteres Mapping auf ein Indexierungsprofil, das auch die Facetten der Suchmaschine der DDB definiert.

KONTAKTINFORMATION

Fraunhofer IAIS NetMedia
Dr. Kai Stalman
Schloss Birlinghoven
53757 Sankt Augustin

<http://www.iais.fraunhofer.de/DDB.html>

<http://www.iais.fraunhofer.de/netmedia.html>

<http://www.iais.fraunhofer.de/>

DOKUMENTVERSIONSHISTORIE

Version 1.0, 15.12.2010, kst, Draft

Version 1.1, 19.01.2011, kst, Korrekturen

Version 1.2, 24.01.2011, kst, Finalisierung

Version 1.3, 01.03.2011, kst, kleinere Textkorrekturen

Version 1.4, 11.04.2011, kst, kleinere Anpassung

Inhalt

1 Vorgaben	7
1.1 Vorgaben	7
1.2 Lizenz	7
1.3 Modelle, Standards, Schnittstellen	8
1.4 Qualitätssicherung	9
2 Funktionale Bestandteile	10
2.1 Harvesting Schnittstellen	10
2.2 Input-Datenaufbereitung (Pre-Ingest)	10
2.3 Datenimport (Ingest)	10
2.4 Datenauslieferung (Access)	11
3 Architektur des Systems	13
3.1 Schichten und Komponenten	13
3.2 Datenfluß	14
3.3 Auth & Auth	17
3.3.1 Konzeption des Auth & Auth Systems	17
3.4 Deployment	19
3.5 Abhängigkeiten der einzelnen DDB Eclipse Entwicklungsprojekte	20
3.6 Aktuelle externe Abhängigkeiten (OS Frameworks und Lizenzen)	21
3.7 Konfigurationen	23
4 Ingest Prozess, Dokumentenablage, API	25
4.1 Ingest	25
4.1.1 Ingest Endpoint	25
4.1.2 Ingest Prozess	25
4.1.3 Binary File Upload	27
4.2 Dokumentenablage (Storagekonzept)	28
4.3 API	29
4.3.1 REST API	29
4.3.2 JavaScript API	34
5 Das Linked Data Konzept	36
5.1 Datenmodell	36
5.1.1 Entities	36
5.1.2 Properties	37
5.1.3 Mapping	38
5.2 Indexierungsprofil	40
5.3 Clustering und Deduplikation	43

1 Vorgaben

1.1 Vorgaben

Für das Projekt gelten folgende Vorgaben und Beschlüsse:

- Als Implementierungssprache soll Java verwendet werden.
- Die Plattform wird modular aufgebaut, damit wesentliche Bestandteile der Architektur ersetzt werden können, andererseits sollen Komponenten so geschnitten sein, dass sie sich leicht auf mehrere Maschinen verteilen lassen.
- Die Architektur soll funktionale Erweiterungen unterstützen.
- Die Verwendung von Standards und offenen Schnittstellen ist wie selbstverständlich anzustreben.
- Literale Informationen werden innerhalb der Plattform stets UTF-8 enkodiert.

1.2 Lizenz

Die DDB soll soweit möglich als Open Source Lösung realisiert werden.

Open Source Software bietet andere, flexiblere Möglichkeiten der Erweiterung, Wartung und Anpassung von Softwarekomponenten als Closed Source Software. Es soll daher konsequent darauf geachtet werden, dass nur nicht-virale, offene Lizenzen zum Einsatz kommen, die eine Änderung des eingebundenen Source Codes erlauben und die Einbindung nicht offener Software nicht verhindern. In dieser Hinsicht ist GPL problematisch und Apache 2.0 eine gute Wahl.

Mit Closed Source Software wird eine Wartung und Erweiterung durch den Hersteller erzwungen. Für das unter hohen Zeitdruck (18 Monate Implementierungszeit) stehende DDB Projekt stellt die Verwendung von Closed Source Software bei Kernkomponenten ein Risiko dar, da jeder Bugfix und jede Erweiterung nur beim Hersteller stattfinden kann und die DDB Entwicklung damit von den Zyklen des Hersteller abhängig wird.

Dazu kommt, dass sich bei Closed Source Komponenten das Problem der Folgekosten (Lizenz und Wartung) verlagert. Da (wie nach jetzigen Vorstellungen geplant) ein Betreiber auch Softwarewartungsaufgaben übernehmen soll, stellen Closed Source Komponenten ein Wartungsproblem dar, dass zunächst dem Kompetenznetzwerk als späterem Betreiber zur Abstimmung übergeben und gemeinsam geklärt werden müsste. Der damit verbundene Prozess würde die Zeitplanungen sprengen.

Schließlich gibt es Überlegungen, für die Weiterentwicklung der DDB eine Community aufzubauen. Dies ist nicht möglich, wenn die Lizenz eine Änderung der Sourcen verbietet.

Aus diesen Gründen sollen die Kernkomponenten der DDB unter eine Apache 2.0 Lizenz gestellt werden. Bei der Auswahl aller für den Systemkern verwendeten Frameworks werden konsequent Lösungen herangezogen, die in entsprechender unschädlicher Weise lizenziert sind.

1.3 Modelle, Standards, Schnittstellen

Die Verwendung von Standards wird angestrebt. Es ist jedoch so, dass es für etliche zentrale Probleme keine verbindlichen Standards gibt. Es wird dann versucht, Lösungen anzubinden, die eine möglichst weitgehende Verbreitung gefunden haben.

Die Anbindungen der Komponenten erfolgt wenn möglich über offene Schnittstellen, etwa HTTP, REST, WSDL, JDBC. Andere offene Standards, die in diesem Projekt eingesetzt werden, sind RDF, CIDOC CRM, JSON, XML, XSL, XACML.

Die Anbindung kommerzieller Basiskomponenten (wie z.B. eine Cloud, ein verteilter Storage, eine kommerzielle Datenbank) ist daher keineswegs ausgeschlossen. Welches Basissystem zum Betrieb sinnvollerweise angeschlossen werden, hängt auch von den Möglichkeiten des Betreibers ab.

Neben den genannten offenen Standards lehnt sich das Konzept der Plattform an folgende Modelle an:

OAIS¹ und **DLRM**²: die DDB implementiert einen kleinen Teil des Modells, insbesondere *Ingest, Access, Storage*. Langzeitarchivierung (Preservation Planning) ist dagegen nicht Bestandteil des Projekts. Das DLRM führt eine Reihe von Domänen zur Modellierung der Entitäten und Funktionen einer Digital Library ein, darunter vor allem Begriffe wie *Resource, Content, User, Functionality, Policy*. OAIS und DLRM definieren Bestandteile einer Nomenklatur, an die sich das DDB Projekt anlehnt.

SOA: die Kernkomponenten der DDB sind als technische Services implementiert, die über Serviceschnittstellen angebunden sind. Die DDB Plattform ist ebenfalls als Service zu verstehen, der mit heterogenen Metadaten ausgezeichnete Objekte in ein RDF-basiertes Modell überträgt, analysiert und vernetzt, ein Indexierungsprofil erzeugt und die berechneten Strukturen über das HTTP Protokoll Maschinen und Menschen für Sichtung (XHTML) und Recherche (RDF, XHTML/RDFa) zugänglich macht. Auf dieser technischen Grundlage können weitere Mehrwertdienste entwickelt werden.

REST und **SOAP** (WSDL) Webservices: die Komponenten der DDB kommunizieren Daten über HTTP und verwenden dabei Webservice Standards.

¹ Reference Model for an Open Archival Information System (OAIS) , <http://public.ccsds.org/publications/archive/650x0b1.PDF>

² The DELOS Digital Library Reference Model, http://www.delos.info/files/pdf/ReferenceModel/DELOS_DLReferenceModel_096.pdf

REST und SOAP Schnittstellen ermöglichen auch die Anbindung von Drittsystemen und Services (Mashups) an die DDB. Die DDB veröffentlicht ein API zum Anschluss anderer Systeme, Komponenten oder Services. Ergänzend zum REST API wird ein Java Binding und ein JavaScript API bereit gestellt.

1.4 Qualitätssicherung

Die Qualität wird in diesem Projekt durch Tests, Reviews und Audits gesichert. Das Entwicklungsteam hat sich darauf verständigt, sämtliche Komponenten durch Unittest und die Plattform durch Integrationstests testbar zu machen.

Die Tests werden bei jedem Checkin über einen Integrationsserver (Hudson) automatisch angestoßen. Bei Fehlern im Build und scheiternden Tests werden die Entwickler per Mail alarmiert.

Neben den Entwicklertests (Unittests) werden Integrationstests geschrieben. Diese Tests stellen sicher, dass die Plattform zu jedem Zeitpunkt funktional korrekt arbeitet.

Metriken zur Codequalität, u.a. die Testabdeckung werden ebenfalls kontinuierlich ausgewertet. Bis Projektende wird eine weitestgehend vollständige Abdeckung aller relevanten Methodenimplementierungen angestrebt.

Zur weiteren Qualitätskontrolle sind Audits vorgesehen. Reviews werden entwicklungsbegleitend durchgeführt.

2 Funktionale Bestandteile

Dieses Kapitel dient der Übersicht und skizziert die funktionalen Bestandteile des Gesamtsystems.

2.1 Harvesting Schnittstellen

Die DDB harvestet andere Systeme. Vorgesehen ist vor allem ein Harvesting über OAI-PMH, außerdem werden Daten über FTP geladen.

Weitere Schnittstellen zu Anbietersystemen, die später umgesetzt werden könnten, wären ATOM (SWORD), SRU, Z39.50. Dafür gibt es zur Zeit keine konkrete Planung.

2.2 Input-Datenaufbereitung (Pre-Ingest)

Die in der Plattform verarbeiteten Daten werden vor dem Ingest für die Plattform aufbereitet.

Die Aufbereitung muss praktisch beliebige Metadaten und Formate in ein geeignetes Modell übertragen und so normalisieren, dass Informationen über die Objekte des kulturellen Erbes in sinnvoller Weise für ein Retrieval genutzt werden können.

Die Eingangsdaten müssen außerdem in Dokumente umgeformt werden, die Menschen (HTML) und Maschinen (RDF) ermöglichen, die Daten zu verstehen.

Die Aufbereitung, bestehend aus der Metadaten transformation und dem Rendering von (statischen) Sichten auf die Daten, wird im folgenden als Pre-Ingest bezeichnet. Es ist ein Werkzeug geplant, das Spezialisten bei der Anpassung der Transformationen unterstützt.

Die Aufbereitung erfolgt unabhängig davon, ob bei einem Ingest auch binäre Inhalte in die DDB Plattform transferiert werden sollen oder nicht.

2.3 Datenimport (Ingest)

Der *Ingest* ist Teil des OAIS Modell und stellt den Prozess der Aufnahme des Submission Information Packages (SIP) und die Umwandlung des SIP ins Archival Information Package (AIP) inklusive der Ablage des AIP im Archival Storage dar. Funktionale Bestandteile des Prozesses sind entsprechend der OAIS Nomenklatur *Receive Submission*, *Quality Assurance*, *Generate AIP*, *Generate Descriptive Information* und *Coordinate Updates*. Diese Funktionen werden innerhalb des Ingest Service der Plattform implementiert.

Ergänzend zu den gerade genannten implementiert der Ingest weitere Funktionen:

- Innerhalb der Plattform müssen Identifier für die Information Objects und Non-Information Objects der Plattform vergeben werden. Diese Identifier ermöglichen Zugriff auf die Objekte. Während des Pre-ingest werden zunächst lokale Identifier für Bindings der Entitäten erzeugt, aus denen sich das Metadatencluster eines Objekts aufbaut. Die lokalen Bindings werden beim Ingest durch persistente Identifier und Bindings ersetzt (siehe auch Kapitel 4.3.2).
- Die einzelnen Entitäten, die (meist implizit) in den Ursprungsdaten referenziert und durch das Mapping auf das Modell der DDB explizit als Entitäten ausgewiesen werden, müssen in der Plattform ebenfalls als Ressourcen angelegt werden.
- Beim Ingest werden die Objekte in einem Triplestore vernetzt.
- Die Transitionen über Properties der Entitäten, die ein Informationsobjekt ausbilden, wird beim Ingest in ein Indexing Profile übersetzt. Das Mapping dieses Indexing Profile ist eine Konfiguration des Ingests und übersetzt die Transitionen in ein Subset von DC Terms³. Damit bildet DC Terms die Obermenge der Facettennamen, die dem Nutzer für das Retrieval als Filter angeboten werden.
- Während des Pre-Ingests oder eventuell auch während des Ingests können Anreicherungen erfolgen. Dafür könnte z.B. ein Integrationspunkt für z.B. Contentus Dienste, etwa für Named Entity Recognition (NER), aber ggf. auch für die automatische Extraktion von Metadaten aus Bildern, Audio- oder Videostreams geschaffen werden.

2.4 Datenauslieferung (Access)

Die im OAIS in der Access Entität zusammengefassten Funktionen sind in der DDB Plattform als zwei Services angelegt: Access für den Zugriff auf Ressourcen im Storage, Search für den Zugriff über die Suchmaschine. Search entspricht bei OAIS den *Query Requests* des Data Managements, das *Results Sets* zur Präsentation an den Nutzer ausliefert.

OAIS kennt außerdem *Report Requests*, die eine Reihe von Abfragen bündeln und formatierte *Reports* an einen Consumer liefern, und *Orders*, die entweder an Data Management oder an Archival Storage (oder beide) gerichtet werden, um ein formales DIP aufzubereiten und auszuliefern. Orders bleiben vorläufig ohne Entsprechung in der DDB. Report Requests werden von Providerseite an die Plattform gestellt, um die Ergebnisse eines Ingests abzufragen (diese werden als Ressourcen unter einer Ingest-Event-Id abgelegt).

Die entsprechenden API Requests werden RESTful ausgeführt.

Search liefert Resultlists und Facetten an einen Client.

³ DCMI Metadata Terms, <http://dublincore.org/documents/dcmi-terms/> und <http://dublincore.org/2010/10/11/dcterms.rdf>

Access liefert Content aus, also Streams, X(HT)ML, RDF (z.B. in XML-Serialisierung) und Json.

3 Architektur des Systems

Das Gesamtsystem besteht aus zwei funktionalen Teilsystemen und einem Authentifizierungs- & Authorisierungssystem.

Die funktionen Teilsysteme sind:

- Augmented SIP Creator (ASC) für den Pre-Ingest
- der DL Core, der die Plattform realisiert, auf der die Deutsche Digitale Bibliothek betrieben wird.

Die Architektur der DDB wird von Komponenten bestimmt, die über einen IoC Container per Dependency Injection (DI) eingebunden werden. Sämtliche Komponenten sind daher gegen Interfaces implementiert. Als Container kommt Spring zum Einsatz.

Bei Systemstart initialisiert der Starter eine von mehreren Spring-Konfigurationen und fährt eine Reihe von Servern hoch. Die Server können je nach gewählter Konfiguration und je nach Deployment auf eine oder mehrere Maschinen oder virtuelle Maschinen verteilt sein. Beim ersten Start werden außerdem die Backends initialisiert.

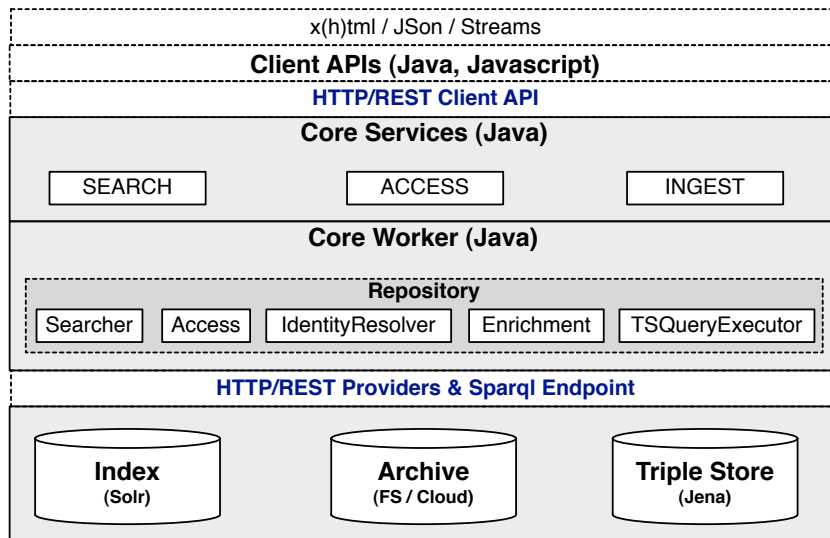
In der aktuellen Version entstehen keine äußeren Abhängigkeiten, d.h. das per Build und Deployment zur Verfügung gestellte System ist self-contained und initialisiert sich automatisch. Eine Ausnahme hiervon bildet das (optionale) Authentifizierungs- und Authorisierungssystem (AAS), das als komplett eigenständige Anwendung nur über Token mit dem Kernsystem kommuniziert.

Die Komponenten der DDB sind auf mehrere Schichten verteilt, diese werden im Folgenden als Erstes beschrieben. In den weiteren Abschnitten kommen Datenfluß, AAS, Deployment und Dependencies zur Darstellung.

3.1 Schichten und Komponenten

Die Präsentationsschicht kommuniziert über REST mit der Plattform. Das REST API (siehe Abschnitt 4.3) definiert, wie Datenobjekte abgefragt und geschrieben werden. Die Datenobjekte werden als XHTML, als JSON bzw. als Streams an den Client geliefert.

Zur einfachen Implementierung von Clients kann ein Java-Binding sowie ein Javascript-Binding, das die Ajaxifizierung übernimmt, genutzt werden.



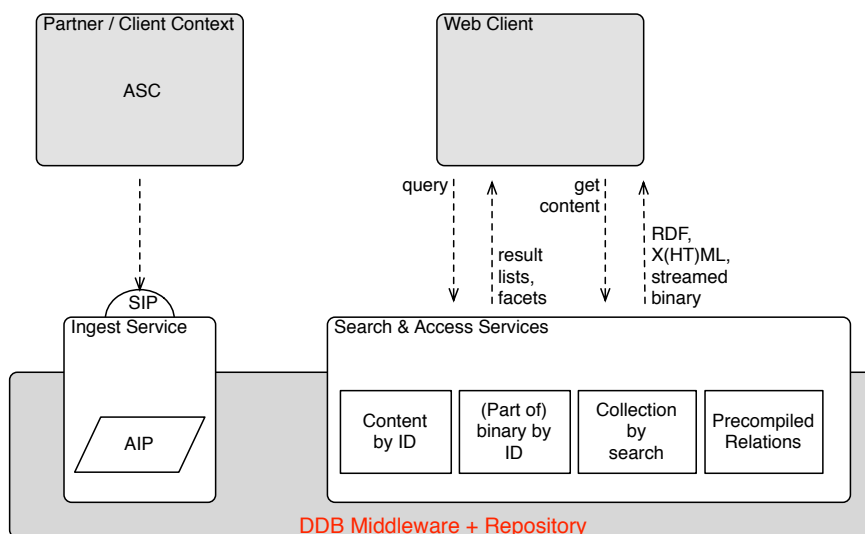
Die obere REST Schnittstelle des Systems wird von einem REST Server implementiert, der die Interfaces der Service Schicht aufruft, dessen Java Implementierungen (Core Worker) als Spring Beans injiziert sind.

Die Worker, in denen die Logik realisiert ist, erhalten Daten als XML bzw. JSon. Um Marshallingkosten zu reduzieren und um schneller Anpassungen am Datenmodell vornehmen zu können, wurde auf eine Modellierung der Daten in der Implementierungssprache Java weitestgehend verzichtet.

Die Worker kommunizieren mit den ebenfalls lose gebundenen Backends (dem Index Server, dem Archive, dem Triple Store) wieder über HTTP (REST, bzw. einen Sparql Endpoint, zur Zeit ist die Joseki Implementierung im Einsatz).

3.2 Datenfluß

Die Ursprungsmetadaten liegen z.B. in Formaten wie Dublin Core, Marc21, MAP, EAD, LIDO, museumdat bei den Einrichtungen vor. Diese Formate werden als Ausgangspunkt des Mappings auf das Modell der DDB akzeptiert.

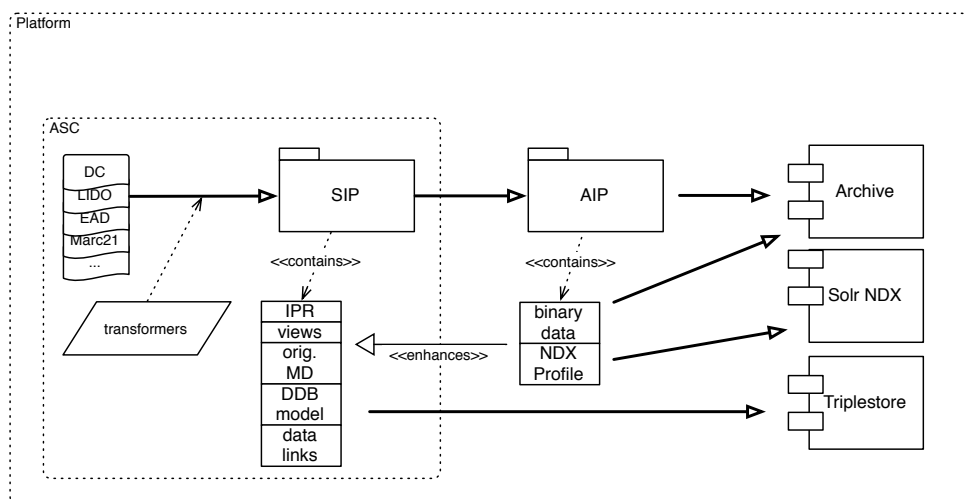


Binäre Daten, Digitalisate oder Derivate, können ebenfalls übertragen werden. Inhalte können Bücher, Bilder, Filme, Audioaufnahmen, Games, Datasets etc sein. Binäre Daten werden während des Ingests ggf. als Streams auf die Plattform heruntergeladen. Die herunterladbaren Binärdaten werden im SIP gekennzeichnet.

Zur Aufbereitung und für den Upload der SIPs dient der ASC (Augmented SIP Creator). Der ASC wird als separate Anwendung beim Datenlieferer bzw. Aggregator oder bei Betreiber bedient. Der ASC prozessiert die Mappingskripte, mit denen die Ursprungsmetadaten in das Datenmodell der DDB übertragen werden. Außerdem stellt der ASC mit Hilfe von Templates statische XHTML Views der einzelnen Objekte aus den Ursprungsmetadaten her. Diese Views werden nach dem Ingest im Storage der Plattform gespeichert und zur Anzeige (Resultlist der Suche, Detailsviews) im Browser verwendet.

Schließlich dient der ASC der Datenqualitätssicherung. Mithilfe von Metriken sollen unbrauchbare Uploads verhindert werden.

Alle Funktionen des ASC sind als (grafisch modellierte) Prozesse organisiert. Die technische Dokumentation des ASC ist nicht Gegenstand dieses Papiers. Für die Plattform erfüllt der ASC den Zweck der Datennormalisierung und sendet die Daten an die Plattform.



Die an die Plattform als SIP gesendeten Daten werden von dem Ingest Service verarbeitet. Der Ingest ist in Abschnitt 4.1 beschrieben. Nach dem Ingest sind die Metadaten indiziert, die SIPs als Archival Information Packages (AIP) im Archiv gespeichert und die Objekte im Tripelstore vernetzt.

Client Applikationen, z.B. ein Webbrowser, stellen über HTTP Suchanfragen an die Plattform. Die Anfragen werden als POSTs über REST ajaxifiziert an die Suchmaschine geleitet. Als Suchmaschine kommt Solr zum Einsatz. Facetten zu den jeweiligen Ergebnislisten werden ebenfalls als POSTs über Ajax Callbacks an den Client übermittelt.

Die Ergebnislisten sind JSON Objekte, die statische HTML Snippets enthalten und über Links auf Detailviews der Objekte verweisen. Die Objektsichten werden per GET abgefragt. Die REST Pfade sind so aufgebaut, dass per

```
GET http://<domain>:<port>/<item-id>
```

auf ein Informationsobjekt zugegriffen werden kann. Die Abfrage einer derartigen URL produziert eine (Default-)Repräsentation des Informationsobjekts.

Die Form der Default-)Repräsentation noch nicht endgültig definiert: vermutlich wird es ein XHTML/RDFa Objekt sein. Neben der Default-Repräsentation des Informationsobjekts gibt es eine Reihe von weiteren Sichten auf das Objekt, die als Komponenten bezeichnet werden. Die Abfrage einer Komponente erfolgt in der Form:

```
GET http://<domain>:<port>/<item-id>[/component]
```

Beispiele für Komponenten sind:

- Description: Konzise literale Beschreibung, geeignet als Label.
- Preview: Kurze Beschreibung des Objekts zur Verwendung in der Suchergebnisliste.
- View: Statische Sicht auf das Objekt.
- RDF: RDF XML Serialisierung des Datenmodells des Objekts.
- Serialisierung der Daten des Indexer Profiles.

sowie andere, zur Zeit noch nicht definierte Sichten.

Eine Besonderheit des Datenflusses ist, dass die im Triplestore abgelegten Daten während des Ingests ausgewertet werden, der Triplestore aber (in der Regel) nicht zur Query-Zeit befragt werden muss, da relevante Transitionen vorberechnet und im Index abgelegt werden.

Schreibende Zugriffe finden immer als Ingest statt. Neben den Massendaten, die über den ASC in die Plattform importiert werden, kann eine Clientanwendung der Plattform ‚native Inhalte‘ erzeugen. Beispiele für native Inhalte sind News, virtuelle Ausstellungen oder About-Pages über Personen oder Einrichtungen. Auf konventionellen Plattformen werden solche Inhalte üblicherweise von einem CMS verwaltet. In der DDB werden alle Inhalte in gleicher Form als linked data Ressourcen verwaltet und in einheitlicher Weise verarbeitet.

3.3 Auth & Auth

Die Plattform kann so konfiguriert werden, dass schreibende und/oder lesende Zugriffe reglementiert werden.

In der aktuellen Implementierung wird darauf verzichtet, Ergebnislisten von Suchabfragen dynamisch auf Rechte zu überprüfen, da wir davon ausgehen, dass nur solche Inhalte indiziert werden, die im öffentlichen Netz sichtbar sein sollen.

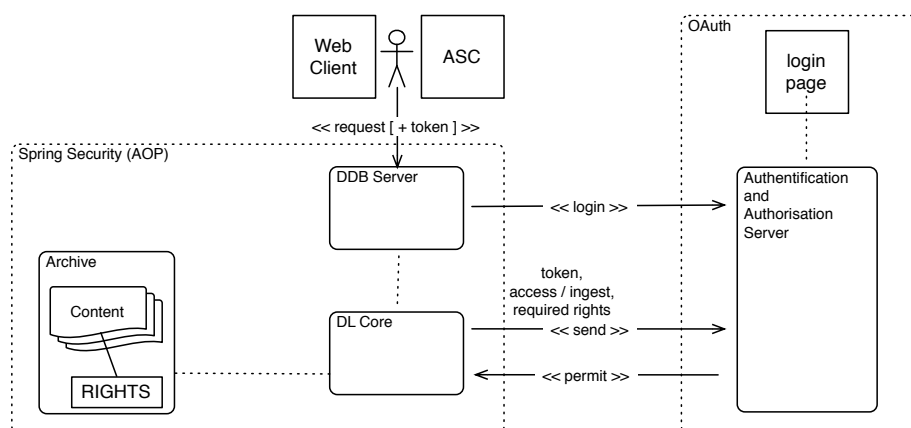
Der lesende Zugriff auf Objekte erfolgt über URLs. Der Zugriff auf eine URL kann reglementiert werden.

Geschützte Digitalisate liegen nach aktuellem Erkenntnisstand beim Provider und damit außerhalb der Verantwortung der DDB Plattform. Prinzipiell können jedoch geschützte Digitalisate auch innerhalb der Plattform so abgelegt werden, dass kein unerlaubter Zugriff möglich ist.

Der Ingest Service der Plattform bietet den einzigen Schreibzugriff der Plattform an. Jeder Ingest erfordert eine Authentifizierung und Authorisierung des Nutzers.

3.3.1 Konzeption des Auth & Auth Systems

Nutzerdaten (Name, Passwort) sollen aus der DDB herausgehalten werden und in einem separaten System (AAS) verwaltet werden. Plattform und AAS kommunizieren über ein Token miteinander.



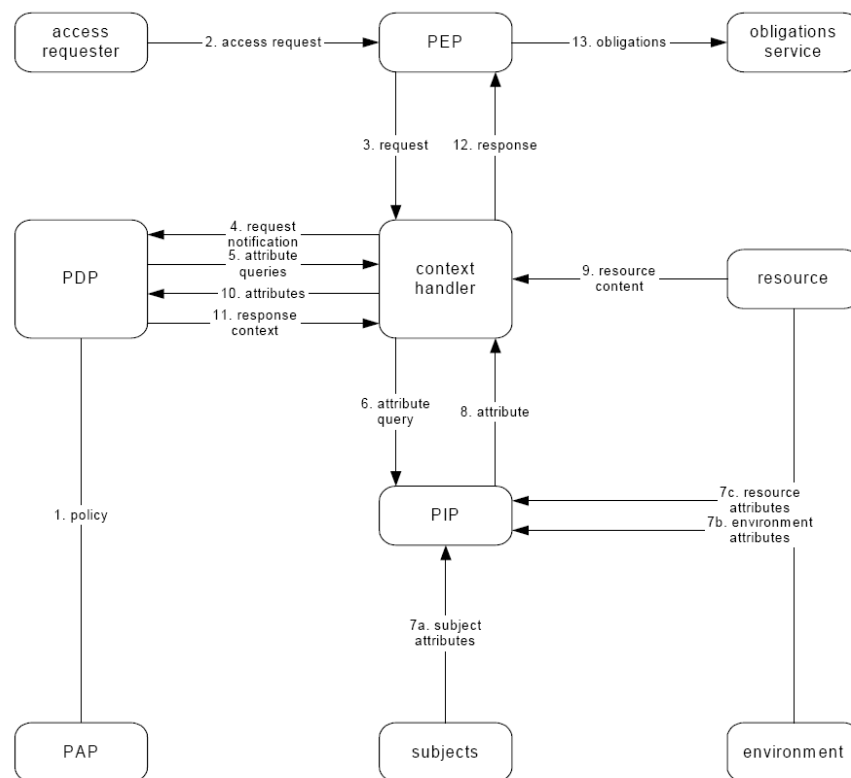
Nutzer erhalten ein Token durch ein Login auf der vom AAS veröffentlichten Login Seite. Nach erfolgreichem Login wird das Token mit dem Request an die DDB Plattform gesendet. Zur Realisierung des Authentifizierungsmechanismus inklusive der Weiterleitung auf eine Login Page wird eine der OAuth Spezifikation (<http://oauth.net/core/1.0/>) entsprechende Implementierung eingesetzt. Dank dieser Indirektion kann die Permission Control aus der Plattform herausgehalten werden.

Innerhalb der DDB Plattform wird der Token per Spring AOP in der Serviceschicht transparent weitergeleitet. Innerhalb der Serviceschicht

entscheidet eine zentraler Permission Provider, ob die angefragte Operation bzw. Resource geschützt ist. Wenn das der Fall ist, wird das Token sowie Details der gewünschten Operation zur weiteren Prüfung an den AAS gesendet.

Die Implementierung des Permission Control Systems des AAS folgt einem OAIS Standard (<http://www.oasis-open.org>), der eXtensible Access Control Markup Language (XAML).

XACML definiert eine Policy Sprache, eine Response und Request Sprache, Datentypen und eine Architektur für eine Permission Control System. Die Architektur stellt sich wie folgt dar:



PEP: Permission Enforcement Point, PIP: Policy Information Point, PDP: Policy Decision Point, PAP: Policy Administration Point.

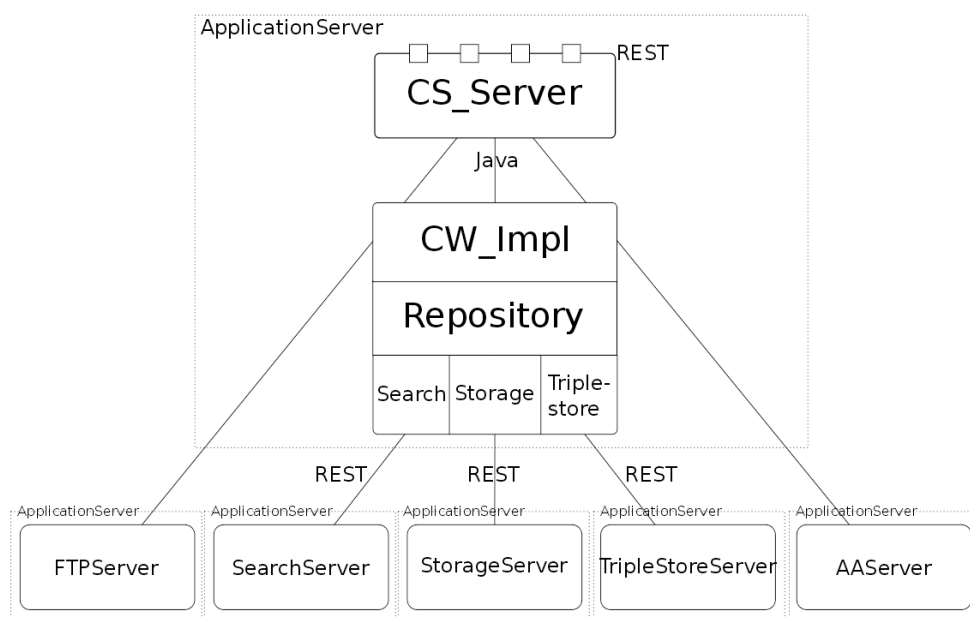
Innhalb des AAS ist der XACML Permission Enforcement Point (PEP) implementiert und entscheidet, ober der Request rechtmäßig ist oder abgelehnt werden muss. Das Ergebnis des Permission requests wird an die DDB Plattform gesendet, die dann die weitere Verarbeitung vornimmt oder die Verarbeitungsanfrage ablehnt.

Ein dieser Architektur entsprechendes System wird gegenwärtig (Januar 2011) in den AAS Prototypen integriert. Die Implementierung des AAS soll beim Betreiber erfolgen.

3.4 Deployment

Die Plattform der DDB wird über einen Applikationsserver per REST von einem Client angesprochen.

Als wichtigste im Rahmen der Projekts realisierte Client-Applikationen sind der ASC und die Web Applikation zu nennen, die das Suchen und Browsen und die Eingabe von Content wie About Pages oder News ermöglicht. Diese letztgenannte Client Komponente hat den Namen Object Discovery (OD) und wird zusammen mit anderen Web Komponenten von einem üblichen Open Source Servlet Container (Jetty) gehostet. Als Templatesprache wird Freemarker verwendet. OD und ASC werden hier nicht weiter diskutiert.



Die Deployment Architektur der DDB Plattform ist so ausgelegt, dass sich einzelne Server Komponenten leicht auf verschiedene Maschinen verteilen lassen.

Der Build Prozess erzeugt, wenn vollständig ausgeführt:

- Webservice und ASC als separate Ziele
- ein Installationspaket für eine einzelne Maschine
- ein Installationspaket für eine verteilte Architektur

Mit der einfachen Ausführung können alle Server nur auf einer Maschine betrieben werden. Die Installation beschränkt sich darauf, das Installationspaket auf die Maschine zu kopieren, ggf. die Ports zu konfigurieren und die Server über dem zentralen Starter hochzufahren.

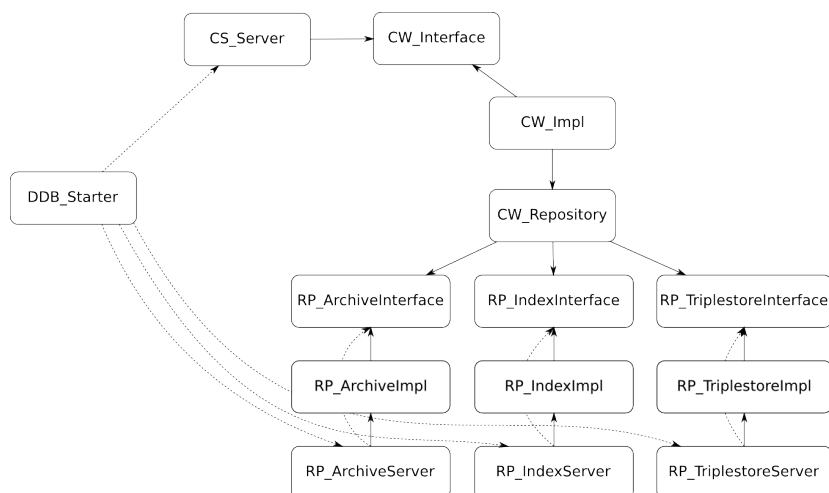
Für eine verteilte Installation werden folgende Pakete erzeugt:

- CSServer: Hostet die REST Schnittstelle der Plattform und die Businesslogik des Systems (Service Schicht).
- IndexServer: kapselt den Index, ist ebenfalls über REST angebunden und verwendet JJson als Transportobjekte. Als Suchmaschine kommt Solr / Lucene zum Einsatz. Andere Suchmaschinen könnten ohne Architekturänderungen alternativ angebunden werden.
- TripleStoreServer: kapselt einen Joseki Endpoint, der einen Triplestore anspricht. Die aktuell verwendete Triplestore Implementierung stammt aus dem Jena Semantic Web Framework (<http://jena.sourceforge.net/>). Für den Produktivbetrieb wird auf ein leistungsfähiges, skalierendes System umgestellt. Der Joseki Endpoint wird von uns zur Zeit refaktoriert (Performanzoptimierung, Konfiguration).
- ArchiveServer: speichert die Objekte der DDB zur Zeit im Filesystem des Servers, kann später durch eine Storage Cloud ersetzt werden.
- FTPServer: dient dazu, den auf dem Webserver ggf. erzeugten nativen Content und den während der Masseningests vom Provider herunterzuladenen binären Content zwischenspeichern.

Der CSServer kann von einem Loadbalancer auf mehrere Maschinen verteilt werden. Der lesende Zugriff verlangt keine sticky sessions. Für den Schreibzugriff muss der Nutzer an einen bestimmten Webserver gebunden werden, wenn er binäre Daten hochlädt.

Die übrigen Server sind entweder von Hause aus selbst verteilbar (Solr) oder sollen bis zum Betrieb durch verteilbare Komponenten ersetzt werden.

3.5 Abhängigkeiten der einzelnen DDB Eclipse Entwicklungsprojekte



Die dargestellten Abhängigkeitsverläufe sollen zeigen, dass beim Systemdesign darauf geachtet wurde, Abhängigkeiten immer nur von oberen zu unteren Schichten verlaufen zu lassen. Implementierende Klassen werden immer gegen Interfaces programmiert.

Gegenwärtig besteht das Gesamtprojekt aus knapp 30 einzelnen Eclipse-Projekten. Das Build verlangt eine Maven Plugin in Eclipse oder kann alternativ von der Console angestoßen werden.

Die an die Interfaces gebundenen Spring Beans werden im Projekt X_Config definiert. Dort befinden sich auch die Spring Properties. Die Namen der Spring Application Context Files wurde so gewählt, dass sie den einzelnen Schichten leicht zuzuordnen sind.

Die Java Interfaces der unteren Schichten (RP_* steht für Repository) werden an zwei Implementierungen gebunden:

- Die RP_*Server (Archive, Index, TripleStore) hosten die REST Ressourcen und reichen die eingehenden Calls lediglich an die Implementierungen weiter. Hier wird der Solr Server und die Joseki Schnittstelle angebunden.
- Die RP_*Impl(ementierungen) realisieren ggf. die Backend-Funktionalität. Für Tests liegen Mock Implementierungen vor. Da Solr und Joseki in der darüber liegenden RP_*Server Schicht direkt angebunden sind, entfallen Implementierungen für Index und TripleStore. Das Filesystem Archive, das gegenwärtig zur Speicherung der Daten genutzt wird, kann ebenfalls entfallen, wenn RP_Archive Server auf eine Storage Cloud verdrahtet ist.

3.6 Aktuelle externe Abhängigkeiten (OS Frameworks und Lizenzen)

Die Projekte verwenden (wie unter Abschnitt 1.2 dargestellt) eine Reihe von Open Source Frameworks. Bei der Auswahl der Frameworks wurden (neben dem Hauptkriterium der passenden Funktionalität drei weitere Kriterien angelegt:

- Bewährt in zahlreichen Projekten (Stabilität)
- Hoher Bekanntheitsgrad in der Entwicklergemeinschaft (leichte Einsetzbarkeit im Projekt)
- Geeignete Lizenz (Apache, MIT License, LGPL oder ähnlich)

Zum Zeitpunkt der Erstellung dieses Dokuments (Januar 2011) ist die Funktionalität der Plattform in weiten Teilen realisiert. Gegenwärtig befinden sich folgende Frameworks im Einsatz:

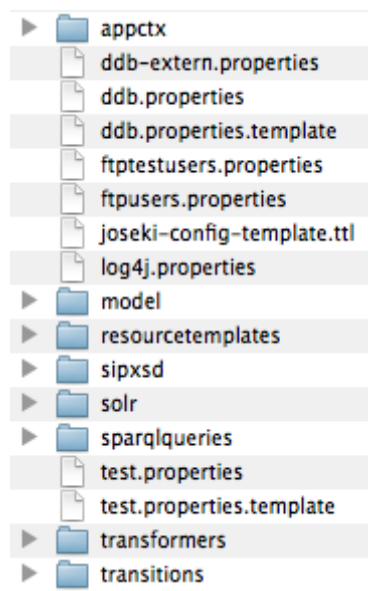
Annotation 1.0	The Apache Software License
AOP alliance	Public Domain
Apache Log4j	The Apache Software License
Apache Solr Core	Apache 2
Apache Solr Solrj	Apache 2
ARQ	BSD-style license
ASM Commons	The Apache Software License
ASM Core	The Apache Software License

ASM Tree	The Apache Software License
Codec	The Apache Software License
Commons FileUpload	The Apache Software License
Commons IO	The Apache Software License
Commons Lang	The Apache Software License
Commons Logging	The Apache Software License
Commons Net	The Apache Software License
COS	Jason Hunter License
Data Mapper for Jackson	The Apache Software License
dom4j	BSD Style
formatJson	Beispielcode / unklar
FreeMarker	BSD-style license
Gson	The Apache Software License
HSQLDB Database	BSD style open source license
HttpClient	Apache License
ICU4J	ICU License
Jackson	The Apache Software License
Java Authentication SPI for Containers	The Apache Software License
java-diff-utils	The Apache Software License
JavaBeans Activation Framework (JAF)	Common Development and Distribution License (CDDL) v1.0
JavaMail API	Common Development and Distribution License (CDDL) v1.0
JAX-RS provider for JSON content type	The Apache Software License
jaxen	Apache Style
JDOM	Weak copyleft
Jena	BSD
Jena IRI	BSD-style license
jersey-client	CDDL 1.1
jersey-core	CDDL 1.1
jersey-server	CDDL 1.1
jersey-spring	CDDL 1.1
Jetty :: Aggregate :: All core Jetty	Eclipse Public License - Version 1.0
Jetty Server	Eclipse Public License - Version 1.0
Jetty Utilities	Eclipse Public License - Version 1.0
jquery	MIT License
jsr311-api	CDDL License
JTA 1.1	The Apache Software License
JUnit	Common Public License Version 1.0
Log4j	The Apache Software License
Lucene Analyzers	Apache 2
Lucene Core	Apache 2
Lucene Highlighter	Apache 2
Lucene Memory	Apache 2
Lucene Miscellaneous	Apache 2
Lucene Queries	Apache 2

Lucene Snowball	Apache 2
Lucene Spellchecker	Apache 2
Mime Detection Utility	The Apache Software License
Servlet Specification API	Apache License Version 2.0
SLF4J API Module	MIT License
SLF4J LOG4J-12 Binding	MIT License
Solr Specific Commons CSV	Apache 2
Spring Framework	The Apache Software License
Spring Framework: AOP	The Apache Software License
Spring Framework: Beans	The Apache Software License
Spring Framework: Context	The Apache Software License
Spring Framework: Core	The Apache Software License
Spring Framework: Web	The Apache Software License
StAX API	The Apache Software License
Streaming API for XML (STAX API 1.0)	The Apache Software License
tiny_mce	LGPL
Woodstox	The Apache Software License
XMLUnit for Java	BSD License
xom	LGPL

3.7 Konfigurationen

Das Verhalten des Systems wird durch eine Reihe von Konfigurationsdateien bestimmt, die in einem zentralen Konfigurationsordner liegen:



Der Ordner `appctx` enthält die Springkonfigurationen. Das Initialisierungsverhalten von Spring wird zusätzlich über Properties gesteuert. Die gewünschte Springkonfiguration kann dem Starter als Argument übergeben werden.

Der Ordner model enthält das Datenmodell der DDB als RDF (gegenwärtig in XML serialisiert).

Der Ordner Transitions enthält die Mappings, mit denen beim Ingest das interne Datenmodell in das Indexing Profile übersetzt wird.

Die weiteren Konfigurationen Solr, den FTP Server und stellen verschiedene Templates bereit.

4 Ingest Prozess, Dokumentenablage, API

4.1 Ingest

Der Ingest bündelt den größten Teil der Intelligenz des Systems.

Der Ingestservice nimmt an einem REST Endpoint die SIPs entgegen, verarbeitet sie und speichert das SIP, binäre Daten und zusätzlich Metadaten und Views im Repository (Archive, Index, Triplestore).

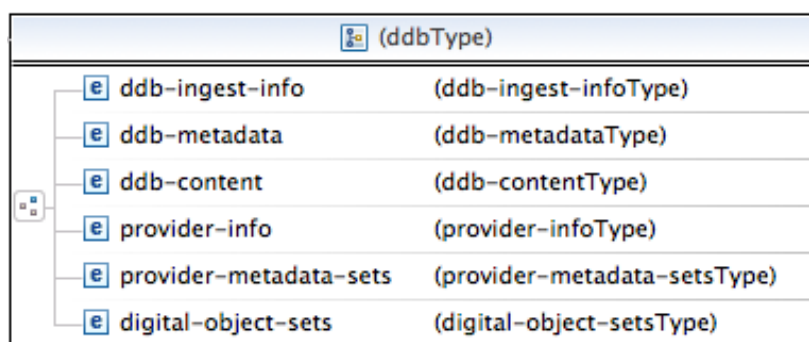
In den folgenden Unterabschnitten wird der Ingest Endpoint, der Ingest Prozess und das Binärdatenhandling beschrieben.

4.1.1 Ingest Endpoint

Der produktive Ingest Endpoint ist eine REST Schnittstelle und damit Teil des REST APIs (siehe auch Abschnitt 4.3).

Der Ingest Endpoint wird über HTTP PUT bedient, der Input ist das SIP. Als Mimetype wird text/xml erwartet.

Die XSD des SIP sieht wie folgt aus:



Die Grafik zeigt nur die Top Level Elemente nach gegenwärtigem Entwicklungsstand (Januar 2011).

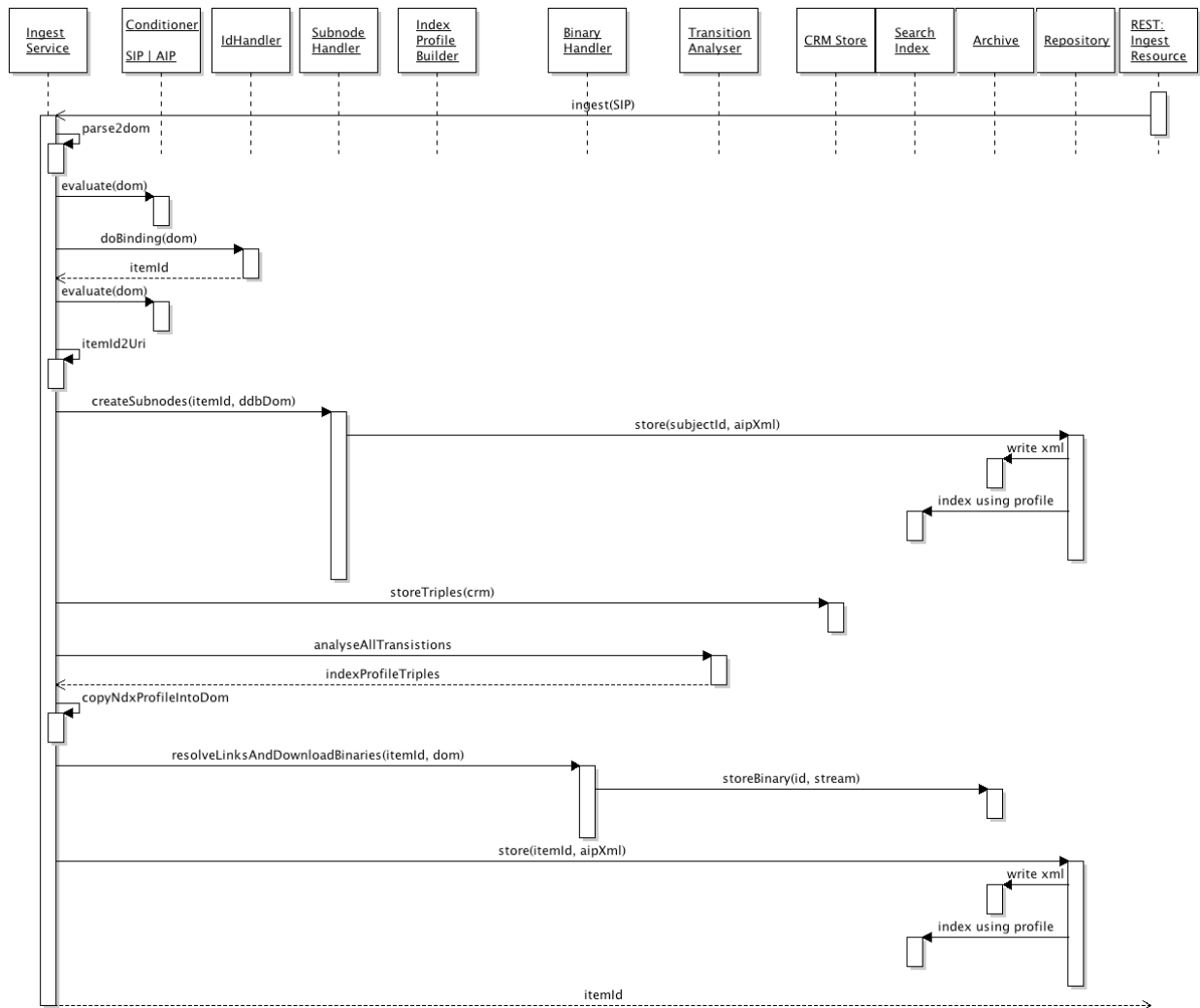
Die jeweils aktuelle, vollständige Fassung kann ausgecheckt unter:

Projekt: X_Config

Pfad: /src/main/resources/sipxsd/sip.xsd

4.1.2 Ingest Prozess

Der Ingest Prozess involviert eine Reihe Services und Subsystemen und läuft wie im folgenden Sequenzdiagramm schematisch dargestellt ab:



Die Ingest Request gehen an der REST Schnittstelle ein und leiten ein SIP in XML Serialisierung an den Ingest Service.

Der **IngestService** validiert und parst den XML Container. Während der nachfolgenden Verarbeitung wird auf einer DOM Repräsentation (jdom) des SIP operiert.

Das Modell enthält lokale Identifier, die im Kontext des jeweiligen SIP eindeutig sind. Die Identifier binden die in der Ontologie beschriebenen Entitäten (z.B. Buchtitel, Museumsobjekt einerseits und andererseits Autor / Creator des primären Objekts. Entitäten dieser Art werden in dem gewählten Repräsentationsmodell im Allgemeinen über Events verbunden, so dass auch die Events als Ressourcen zu betrachten sind und über lokale Identifier verfügen (vergleiche dazu Abschnitt 5.1). Sämtliche, im SIP enthaltenen lokale Identifier werden von einem **IdHandler** aufgelöst und in persistente Identifier übersetzt. Im Zuge der Analyse des Objektmodells wird auch der persistente Identifier des (immer bezogen auf des SIP) primären Informationsobjekts berechnet. Nachdem die lokale Identifier aufgelöst und die persistenten Identifier in den Dom Tree eingetragen sind, bezeichnen wir das SIP als AIP.

Der IngestService bietet die Möglichkeit, vor und nach dem Aufruf an den IdHandler einen **Normalizer** aufzurufen, der einen ICortexModelDomWorkerimplementiert und beliebige Operationen auf dem Datenmodell ausführen kann. Die beiden **Normalizer** bekommen über Spring eine Anzahl von Commands injiziert, die nacheinander abgearbeitet werden (chain-of-command Pattern) und für Transformationen, Normalisierungen, Qualitätssicherungen und Anreicherungen gedacht sind.

Nachdem nun die Identifier feststehen, die Daten normalisiert und angereichert sind, müssen die an das primäre Informationsobjekt gebundenen Entitäten analysiert werden. Diese Entitäten müssen als Ressourcen zur Verfügung stehen und werden, sofern sie nicht bereits existieren, vom **SubnodeHandler** erzeugt.

Die im Modell abgebildeten, vom primären Informationobjekt ausgehenden Transitionen werden mit einer Regelsprache auf ein Indexing Profile gemappt. Der **TransitionAnalyser** liest die Übersetzungsregeln aus einer Registry, die über Spring konfiguriert ist. Nach Abschluss der Transitionsanalyse liegt ein Indexprofil vor, das in den Dom Tree einkopiert wird.

Der **BinaryHandler** analysiert den Teil des Dom, der Links auf herunterladbare binäre Inhalte enthält. Zum Download wird per Default das FTP Protokoll verwendet (andere Protokolle können als Plugin hinzugefügt werden). Der BinaryHandler legt die Streams als Files im Archive ab.

Der letzte Schritt des Ingests besteht darin, das AIP zu speichern und den Index per Indexing Profiles zu aktualisieren. Das AIP wird vor der Ablage in einzelne Dateien zerlegt, so dass Komponenten entstehen, die über REST Pfade direkt angesprochen werden können, ohne dass jeweils das gesamte AIP geparkt werden müßte.

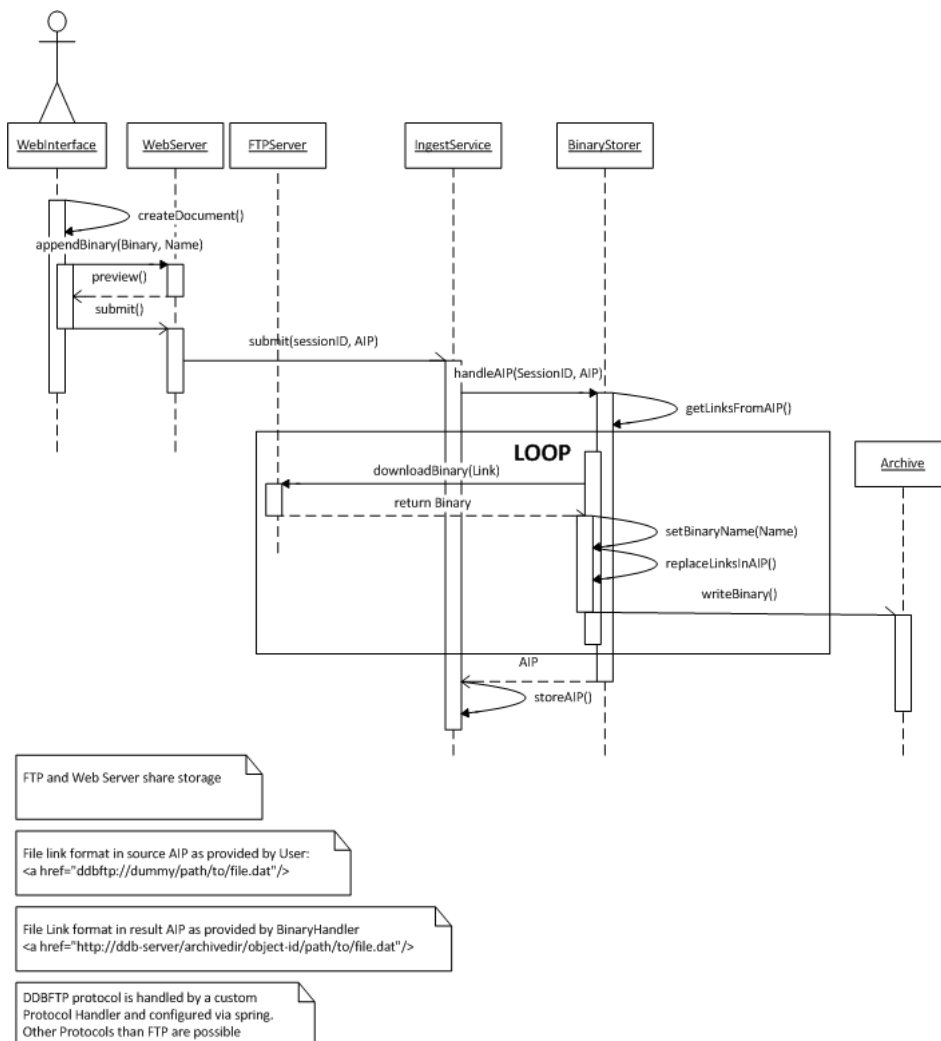
4.1.3 Binary File Upload

Quellen von Binärdaten sind die Web Applikation und der ASC. Zur Übertragung von Binärdaten müssen im SIP Links gespeichert sein, die während des Ingests (siehe Abschnitt 4.1) heruntergeladen werden, da es angesichts der Datenmengen nicht sinnvoll wäre, die Binärdaten als XML Attachements bzw. per Mtom zu serialisieren.

Binärdaten können in der Web Applikation z.B. dann benötigt werden, wenn eine About Page mit einem Bild oder Video erzeugt werden soll. Das WebInterface bietet dem Nutzer den Upload des Files an. Im Kontext der Applikation kann die Datei lediglich auf dem Webserver temporär gelagert werden, da der Ingest ein vollständiges SIP erfordert, das zum Zeitpunkt des Upload nicht zur Verfügung steht, sondern erst dann erzeugt wird, wenn der Benutzer das gesamte Objekt zur Versendung freigibt.

Das SIP wird so erzeugt, dass die hochgeladenen Binärdateien als Links auf ein temporäres Benutzerverzeichnis eingebunden werden. Für die Masseningests wird entsprechend verfahren, denn das SIP enthält einen Knoten mit Links auf Dateien, die über das Internet heruntergeladen werden sollen.

Wenn der Ingest Service das SIP erhält, können die Binaries per FTP vom Webserver bzw. von der Einrichtung herunter geladen werden. Der Vorgang wird im folgenden Sequenzdiagramm detailliert dargestellt:



4.2 Dokumentenablage (Storagekonzept)

Die Dokumentenablage ist eine Aufgabe der OAI Entität *Archival Storage*. Dokument ist hier im allgemeinsten Sinn das im SIP enthaltene XML inklusive aller Content Information und Context Information.

Aus Gründen der besseren Performanz ist es sinnvoll, die AIPs redundant so zu speichern, dass die gespeicherten Einheiten den Abfragen entsprechen.

Nach Prüfung verschiedener Optionen wurde auf die Verwendung eines üblichen Asset Management System wie Dspace, Fedora Commons, CDS Invenio, Eprints, MyCoRe oder OPUS verzichtet.

Statt dessen wird für die Produktionsumgebung eine (private) Storage Cloud priorisiert, die bezüglich der abgelegten Einheiten mit dem aktuellen Dateisystemspeicher kompatibel ist.

Momentan wird ein AIP als Verzeichnis abgelegt, das über den Identifier des Informationobjekts anzusprechen ist und die Metadaten, RDF Daten, Views und Binaries als Dateien enthält. Die Restpfade werden auf diese Ressourcen gemappt.

4.3 API

Die Plattform bietet für Ingest, Queries und Access auf Ressourcen drei APIs an:

- das REST API: nutzt HTTP, XML, Json, Streams
- ein intuitives Java Binding zur Verwendung in Servlets oder Java Applikationen, nutzt intern das REST API. *Die Java API Dokumentation (JavaDoc) liegt in den Sourcefiles vor, kann ausgecheckt werden und ist hier nicht dargestellt.*
- ein Java Script AJAX Binding zur Verwendung bei der Website Programmierung: nutzt intern das REST API

4.3.1 REST API

Die REST Calls bestehen aus einem Pfad, der einen als component bezeichnetes Element enthält.

4.3.1.1 ,Components' im REST Pfad

Die CS_Server.* API Calls stellen das aktuelle öffentliche API dar. Die übrigen Calls beschreiben die internen APIs der Plattform.

Als Komponenten (Argument {component}) sind bislang definiert:

- description (String): Titel (Label) eines Objekts
- preview (XHTML, statisch): wird in der Ergebnisliste des Suche angezeigt
- view (XHTML, statisch): wird in der Detailsicht angezeigt. Content stammt aus Originalmetadaten, direkte Einflussmöglichkeit der Einrichtung darauf, was angezeigt wird.
- profile (XHTML+RDFa, Indexing Profile, DC Terms, bei Ingest berechnet, default component): View mit RDF mit den Daten, die für das Indexing Profile aus berechnet wurden.
- rdfa (XHTML+RDFa, dynamisch): Model des primären Objekts, gerendet als XHTML view mit RDF Annotationen.

- rdf2 (RDF aus NDX Profile, dynamisch, Serialisierung tbd.): RDF Repräsentation des Indexing Profiles (DC Terms), für die maschinelle Verwendung.

4.3.1.2 REST Pfade

Stand Januar 2011, die aktuelle Version findet sich unter:
<https://wiki.d-nb.de/display/DDB/DLCore-Services+REST-API>

CS_Server.AccessResource

Method	Pfad	Input	Output	Kommentar
GET	{id}/{component}		text/xml	Hole einzelne Komponente eines AIPs, Ergebnis ist ein (X)HTMLT oder Json Objekt oder ein Binary {component} sollte enum sein oder entsprechend behandelt werden (404)

CS_Server.SearchResource

Method	Pfad	Input	Output	Kommentar
POST	search	application/json <pre>{ "query":"test", "offset":0, "rows":10 }</pre> offset und rows sind optional	application/json <pre>{ "results":[{"id":"101","preview":"Preview"}, {"id":"102","preview":""}], "numberOfResults":2 }</pre>	Suche nach query, schränke Ergebnisse auf rows beginnend mit offset ein
POST	search/facets/build	application/json <pre>{ "query":"test", "minDocs":1, "facets": [{"field":"date"}, {"field":"title"}] }</pre>	application/json <pre>{ "field":"title", "facetValues": [{"value":"Alpina", "count":22}, {"value":"Tabula", "count":1},], "numberOfFacets":2 }</pre>	Gibt alle Facetten der query mit mindestens minDocs Treffern zurück. Facetten werden auf field eingeschränkt. field kann wiederholt auftreten
POST	search/facets	application/json	application/json	FacettenListe als Post

		<pre>{ "query": "test", "offset": 0, "rows": 10, "facets": [{ "field": "date", "facetValues": [{ "value": "1809"}, { "value": "1810"}] }] }</pre> <p>offset und rows sind optional</p>	<pre>{ "results": [{ "id": "101", "preview": "Preview" }, { "id": "102", "preview": "" }], "numberOfResults": 2 }</pre>	
--	--	--	---	--

CS_Server.Ingester

Method	Pfad	Input	Output	Kommentar
PUT	/	text/xml		Speichert Sip

CS_Server.QueryExecutorResource

Method	Pfad	Input	Output	Kommentar
POST	sparql/select/	<pre>application/json { "sparql": " PREFIX foaf: <http://xmlns.com/foaf/0.1/> SELECT ?nameX ?nameY ?nickY WHERE { ?x foaf:knows ?y ; foaf:name ?nameX . ?y foaf:name ?nameY . OPTIONAL { ?y foaf:nick ?nickY } }" }</pre>	<pre>application/json { "head": { "vars": ["book" , "title"] }, "results": { "bindings": [{ "book": { "type": "uri", "value": "http://example.org/book/book6" }, "title": { "type": "literal", "value": "Harry Potter and the Half- Blood Prince" } }, { "book": { "type": "uri", "value": "http://example.org/book/book5" }, </pre>	Ermöglicht das Absetzen einer SPARQL-Query.

			<pre> "title": { "type": "literal" , "value": "Harry Potter and the Order of the Phoenix" } }, { "book": { "type": "uri" , "value": "http://example.org/book/book4" }, "title": { "type": "literal" , "value": "Harry Potter and the Goblet of Fire" } }, { "book": { "type": "uri" , "value": "http://example.org/book/book3" }, "title": { "type": "literal" , "value": "Harry Potter and the Prisoner Of Azkaban" } }, { "book": { "type": "uri" , "value": "http://example.org/book/book2" }, "title": { "type": "literal" , "value": "Harry Potter and the Chamber of Secrets" } }, { "book": { "type": "uri" , "value": "http://example.org/book/book1" }, "title": { "type": "literal" , "value": "Harry Potter and the Philosopher's Stone" } }] } </pre>	
POST	sparql/ask/	<pre> application/json { "sparql": " PREFIX foaf: <http://xmlns.com/foaf/0.1/> ASK { ?x foaf:name "Alice" }" } </pre>	<pre> application/json { "head": {} "boolean" : true } </pre>	Ermöglicht das Absetzen einer SPARQL-Query.

POST	sparql/construct/	<pre> application/json { "sparql": " PREFIX foaf: <http://xmlns.com/foaf/0.1/> PREFIX vcard: <http://www.w3.org/2001/vcard- rdf/3.0#> CONSTRUCT { <http://example.org/person#Alice> vcard:FN ?name } WHERE { ?x foaf:name ?name }" } </pre>	text/xml	Ermöglicht das Absetzen einer SPARQL-Query.
POST	sparql/describe/	<pre> application/json { "sparql": " PREFIX foaf: <http://xmlns.com/foaf/0.1/> DESCRIBE ?x WHERE { ?x foaf:mbox <mailto:alice@org> }" } </pre>	text/xml	Ermöglicht das Absetzen einer SPARQL-Query.

RP_Search.IndexResource

Method	Pfad	Input	Output	Kommentar
PUT	index/{id}/metadata	text/xml		Add CRM metadata to index for document {id}
PUT	index/{id}/preview	text/xml		Add Preview to index for document {id}
PUT	index/{id}/view	text/xml		Add view to index for document {id}
PUT	index/commit			Commit all added documents

RP_Search.SearchResource

Method	Pfad	Input	Output	Kommentar
POST	search	application/json	application/json	Suche nach query, schränke Ergebnisse auf rows beginnend mit offset ein

POST	search/facets/build	application/json	application/json	gibt alle Facetten der query mit mindestens minDocs Treffern zurück. Facetten werden auf field eingeschränkt. field kann wiederholt auftreten
POST	search/facets	application/json	application/json	FacettenListe als Post

RP_Archive.StorageResource

Method	Pfad	Input	Output	Kommentar
GET	{path: .+}			getBinaryItem
GET	{path: .+}		text/xml, text/plain, text/html	getTextItem
PUT	{path: .+}	application/octet-stream	text/plain	createBinaryItem
PUT	{path: .+}	text/xml, text/plain, text/html	text/plain	createTextItem
DELETE	{path: .+}			deleteItem
POST	{path: .+}	text/xml, text/plain, text/html		updateTextItem
POST	{path: .+}	application/octet-stream		updateBinaryItem

4.3.2 JavaScript API

Das JavaScript API wird in einer Web Application eingesetzt. Die aktuell unterstützten Funktionen sind nachfolgend beschrieben:

```
function access(id, component)
Holt die angegebene Komponente eines AIP aus dem Repository.
@param id: ID des AIP
@param component: Komponente des AIP (Titel, Preview, View)
@return: Die gesuchte Komponente des AIP.
```

```
function search(jsonIn)
Durchsucht das Repository und liefert die Resultate von
<code>offset</code> bis <code>offset + rows</code>
@param jsonIn: z.B. { "query":"test", "offset":0, "rows":10 }
@return: Liefert einen Json-String mit den gesuchten Datensätzen.
```

```
function getFacets(jsonIn)
Ermittelt die Facetten einer Anfrage.
```

@param jsonIn: z.B. { "query":"test", "minDocs":1 }
@return: Liefert einen Json-String mit einer Liste der Facetten.

function searchFaceted(jsonIn)
Ermittelt die zu den angegebenen Facettenausprägungen passenden Datensätze.
@param jsonIn: z.B. { "query":"test", "offset":0, "rows":10, "facets":[{ "field":"date", "facetValues": [{"value":"1809"}, {"value":"1810"}] }] }
@return: Liefert einen Json-String mit den gesuchten Datensätzen.

function store(xml)
Speichert ein SIP im Repository.
@param xml: Die Daten des SIP
@return: -

Nicht öffentlich:
function sparqlSelect(sparqlQuery)
Stellt eine SPARQL-Select-Anfrage an den Tripplestore.
@param query: Sparql-Query (JSON)

5 Das Linked Data Konzept

Das Kapitel skizziert die Datenmodellierung und die Vernetzung der Objekte in der Plattform. Das Datenmodell bildet die Grundlage einerseits des Indexierungsprofils und andererseits der Verfahren zur Berechnung von Ähnlichkeitsmaßen, die für die in Release 2 geplanten Deduplikations- und Clustering-Ansätze benötigt werden.

5.1 Datenmodell

Als Datenmodell der DDB wurde das CIDOC CRM⁴ ausgewählt und erweitert, bzw. vor allem mit dem Europeana Data Model (EDM) harmonisiert. Im folgenden wird das Standard CRM skizziert. Die Harmonisierung ist noch nicht abgeschlossen und erfolgt in enger Zusammenarbeit Dr. Martin Doerr⁵.

Das CRM beruht auf zwei sehr übersichtlichen Hierarchien von Entitäten und Properties. Dessen ungeachtet erlaubt das CRM ein hohes Maß an semantischer Präzision. Es eignet sich daher als Zwischenformat, dessen Verwendung die Anzahl der notwendigen Mappings dramatisch reduziert, wenn verschiedene Eingangsformate und mehrere Zielsprache benötigt werden.

5.1.1 Entities

E1	CRM Entity
E2	- Temporal Entity
E4	- - Period
E5	- - - Event
E7	- - - - Activity
E11	- - - - - Modification
E12	- - - - - - Production
E13	- - - - - - Attribute Assignment
E65	- - - - - - Creation
E63	- - - - - - Beginning of Existence
E12	- - - - - - <i>Production</i>
E65	- - - - - - Creation
E64	- - - - - - End of Existence
E77	- Persistent Item
E70	- - Thing
E72	- - - Legal Object
E18	- - - - Physical Thing
E24	- - - - - Physical Man-Made Thing

⁴ CIDOC Conceptual Reference Model, Version 5.0.2, edited by: Nick Crofts, Martin Doerr, Tony Gill, Stephen Stead, Matthew Stiff; http://www.cidoc-crm.org/official_release_cidoc.html

⁵ Das CRM wurde von Dr. Martin Doerr entwickelt und hat sich in den letzten mehr als zehn Jahren in verschiedensten Projekten auch im Bereich des kulturellen Erbes als Modell zur Abbildung semantischer Beziehungen bewährt. Martin Doerr ist research director am Information Systems Laboratory und leitet das Centre for Cultural Informatics am Institute of Computer Science, FORTH.

E90	- - - -	Symbolic Object
E71	- - -	Man-Made Thing
E24	- - - -	Physical Man-Made Thing
E28	- - - -	Conceptual Object
E89	- - - - -	Propositional Object
E30	- - - - -	Right
E73	- - - - -	Information Object
E90	- - - - -	Symbolic Object
E41	- - - - -	Appellation
E73	- - - - -	Information Object
E55	- - - - -	Type
E39	- -	Actor
E74	- - -	Group
E52	-	Time-Span
E53	-	Place
E54	-	Dimension
E59		Primitive Value
E61	-	Time Primitive
E62	-	String

5.1.2 Properties

Jedes Property weist eine eindeutige Kombination von Domain und Range auf.

Property Id	Property Name	Entity – Domain	Entity - Range
P1	is identified by (identifies)	E1 CRM Entity	E41 Appellation
P2	has type (is type of)	E1 CRM Entity	E55 Type
P3	has note	E1 CRM Entity	E62 String
P4	has time-span (is time-span of)	E2 Temporal Entity	E52 Time-Span
P7	took place at (witnessed)	E4 Period	E53 Place
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
P16	- used specific object (was used for)	E7 Activity	E70 Thing
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
P92	- brought into existence (was brought into existence by)	E63 Beginning of Existence	E77 Persistent Item
P108	- - has produced (was produced by)	E12 Production	E24 Physical Man-Made Thing
P94	- - has created (was created by)	E65 Creation	E28 Conceptual Object
P93	- took out of existence (was taken out of existence by)	E64 End of Existence	E77 Persistent Item
P15	was influenced by (influenced)	E7 Activity	E1 CRM Entity
P16	- used specific object (was used for)	E7 Activity	E70 Thing
P20	had specific purpose (was purpose of)	E7 Activity	E7 Activity
P43	has dimension (is dimension of)	E70 Thing	E54 Dimension
P46	is composed of (forms part of)	E18 Physical Thing	E18 Physical Thing
P59	has section (is located on or within)	E18 Physical Thing	E53 Place
P67	refers to (is referred to by)	E89 Propositional Object	E1 CRM Entity
P75	possesses (is possessed by)	E39 Actor	E30 Right
P81	ongoing throughout	E52 Time-Span	E61 Time Primitive
P82	at some time within	E52 Time-Span	E61 Time Primitive
P89	falls within (contains)	E53 Place	E53 Place
P104	is subject to (applies to)	E72 Legal Object	E30 Right
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
P127	has broader term (has narrower term)	E55 Type	E55 Type
P128	carries (is carried by)	E24 Physical Man-Made Thing	E73 Information Object
P130	shows features of (features are also found on)	E70 Thing	E70 Thing
P140	assigned attribute to (was attributed by)	E13 Attribute Assignment	E1 CRM Entity

Property Id	Property Name	Entity – Domain	Entity - Range
P141	assigned (was assigned by)	E13 Attribute Assignment	E1 CRM Entity
P148	has component (is component of)	E89 Propositional Object	E89 Propositional Object

5.1.3 Mapping

Das CRM erlaubt Mappings von beliebigen Ausgangssprachen. Für einige der relevanten Sprachen liegen generische Standardmappings vor, die von den Entwicklern des CRM veröffentlicht wurden.

Die im Rahmen des Entwicklungsprojekts entworfenen Mappings für DC, EAD, LIDO und andere Sprachen orientieren sich an diesen Empfehlungen.

Das Grundprinzip des CRM - und das macht das CRM als Zwischensprache besonders interessant -, besteht darin, die einzelnen in den Originalmetadaten vorgefundenen verschiedenen Entitäten, die ein Objekt ausmachen (z.B. bei einem Buch: der Autor, der Publisher), über entsprechende Events zu vermitteln.

Konkret bedeutet das, dass zwischen Autor und Buch ein CreationEvent steht. Die Events ergeben interessante Kandidaten für eine Deduplikation: zwei identische CreationEvents verweisen offensichtlich auf zwei identische Bücher. Auch wenn diese Bücher von verschiedenen Einrichtungen eingebracht werden und uneinheitlich erfasst wurden, besteht eine Chance, die Identität anhand der Events zu erkennen.

Eine Konsequenz der Verwendung von Modellen wie dem CRM ist, dass die semantische Beschreibung der Objekte, die sich bei den Originalmetadaten mehr oder weniger stark an der Metapher des Bibliothekskatalogs bzw. des Karteikastens orientiert, in eine graphbasierte Darstellung übertragen wird.

Für die Übertragung der flachen Formate in den CRM Graphen werden Sets von Feldern auf Cluster gemappt, wobei die Gesamtmenge der Cluster ein komplettes Mapping ergibt. Für diese Art der Übertragung gibt es die bereits erwähnten konzeptionellen Vorlagen.

Wenn das Ausgangsformat beispielsweise DC ist, entstehen Cluster wie das unten abgebildete.⁶

⁶ DC.type mapping to CIDOC/CRM von Konstantia Kakali, Martin Doerr, Christos Papatheodorou Thomais Stasinopoulou, 2007, http://www.cidoc-crm.org/docs/WP5-T5_5-DC2CRMmapping-060728v0_2-final.doc.

5.2 Indexierungsprofil

Das Indexing Profile der Plattform wird aus den im CRM Model für die einzelnen primären Informationsobjekte erzeugten Relationen und Transitionen berechnet und verwendet das DC Terms Vokabular⁷.

Das Mapping von CRM auf DC Terms ist eine Konfiguration der Plattform und kann auf konkrete Bedürfnisse leicht angepasst werden.

Aus den DC Terms werden die Facetten der Suche erzeugt. Alle im Indexing Profile enthaltenen Werte werden indiziert. Zur Erzeugung der Facetten wird ein zusätzlicher Filter, der ebenfalls konfiguriert werden kann, geschaltet.

http://purl.org/dc/terms/creator	Creator	An entity responsible for making the resource.
http://purl.org/dc/terms/created	Date Created	Date of creation of the resource.
http://purl.org/dc/terms/subject	Subject	The topic of the resource. Typically, the subject will be represented using keywords, key phrases, or classification codes. Recommended best practice is to use a controlled vocabulary.
http://purl.org/dc/terms/format	Format	The file format, physical medium, or dimensions of the resource. Examples of dimensions include size and duration. Recommended best practice is to use a controlled vocabulary such as the list of Internet Media Types
http://purl.org/dc/terms/contributor	Contributor	An entity responsible for making contributions to the resource. Examples of a Contributor include a person, an organization, or a service.
http://purl.org/dc/terms/publisher	Publisher	An entity responsible for making the resource available.

http://purl.org/dc/terms/type	Type	The nature or genre of the resource. Recommended best practice is to use a controlled vocabulary such as the DCMI Type Vocabulary [DCMITYPE].
---	-------------	---

http://purl.org/dc/terms/source	Source	A related resource from which the described resource is derived.
---	---------------	--

⁷ <http://dublincore.org/2010/10/11/dcterms.rdf>

http://purl.org/dc/terms/coverage	Coverage	The spatial or temporal topic of the resource. Temporal topic may be a named period, date, or date range. Recommended best practice is to use a controlled vocabulary such as the Thesaurus of Geographic Names [TGN]. Where appropriate, named places or time periods can be used in preference to numeric identifiers such as sets of coordinates or date ranges.
http://purl.org/dc/terms/rights	Rights	Information about rights held in and over the resource.
http://purl.org/dc/terms/tableOfContents	Table Of Contents	A list of subunits of the resource.
http://purl.org/dc/terms/abstract	Abstract	A summary of the resource
http://purl.org/dc/terms/isPartOf	Is Part Of	A related resource in which the described resource is physically or logically included.
http://purl.org/dc/terms/isReferencedBy	Is Referenced By	A related resource that references, cites, or otherwise points to the described resource.
http://purl.org/dc/terms/educationLevel	Audience Education Level	A class of entity, defined in terms of progression through an educational or training context, for which the described resource is intended.
http://purl.org/dc/terms/relation	Relation	A related resource.
http://purl.org/dc/terms/rightsHolder	Rights Holder	A person or organization owning or managing rights over the resource.
http://purl.org/dc/terms/Location	Location	A spatial region or named place
http://purl.org/dc/terms/PeriodOfTime	Period of Time	An interval of time that is named or defined by its start and end dates.
http://purl.org/dc/terms/temporal	Temporal Coverage	Temporal characteristics of the resource.

http://purl.org/dc/terms/description	<i>Description</i>	Description may include but is not limited to: an abstract, a table of contents, a graphical representation, or a free-text account of the resource.
http://purl.org/dc/terms/title	<i>Title</i>	A name given to the resource.
http://purl.org/dc/terms/identifier	<i>Identifier</i>	An unambiguous reference to the resource within a given context. Recommended best practice is to identify the resource by means of a string conforming to a formal identification system.
http://purl.org/dc/terms/language	<i>Language</i>	A language of the resource Recommended best practice is to use a controlled vocabulary such as RFC 4646 [RFC4646].
http://purl.org/dc/terms/alternative	<i>Alternative Title</i>	An alternative name for the resource.
http://purl.org/dc/terms/URI	<i>URI</i>	The set of identifiers constructed according to the generic syntax for Uniform Resource Identifiers as specified by the Internet Engineering Task Force.

5.3 Clustering und Deduplikation

Wird erst in Release 2 adressiert.