Hochschule für Technik, Wirtschaft und Kultur Leipzig

Semantic Web

Transformation des semantischen Professorenkatalogs Leipzig

Simeon Ackermann

29. Juli 2016

Betreuer: Prof. Dr. Thomas Riechert

Inhaltsverzeichnis

1 Einführung

Die vorliegende Ausarbeitung beschreibt die Vorgehensweise und Implementierung zur Transformation der Daten des semantischen Professorenkatalogs Leipzig[1], basierend auf eine Ontologie von 2013, auf das aktuelle Modell der Ontologie 2016.

Die Ausgangsdaten des Katalogs liegen als SQL Dump einer älteren OntoWiki Instanz vor, wobei alle Tripel in einer Tabelle mit Spalten für Subjekt, Prädikat und Objekt eindeutig erfasst sind. Dieses Projekt analysiert das den Daten zugrunde liegende Modell, erstellt ein Mapping auf das neue Modell und überführt die Daten in das neue Modell als RDF Ausgabe-Format. Dabei wird darauf geachtet, die Lösung möglichst generisch und wiederverwendbar zu realisieren um Basis für ähnliche Transformationsprojekte zu sein.

2 Analyse

Für die Realisierung der Transformation wurde das Modell CPM (*Catalogus Professorum Model*) das alten Professorenkatalogs sowie die Daten zunächst umfassend analysiert um die Zusammenhänge und Beziehungen zu verstehen. Das genaue Verständnis von Modell und Daten wurden hoch priorisiert, um die Transformation möglichst eindeutig und fehlerfrei zu realisieren.

Bei der Analyse wurde eine Divergenz zwischen Daten und Modell festgestellt. So werden Properties und Relationen in Klassen verwendet, die im Modell nicht vorgesehen sind.

[1] http://www.uni-leipzig.de/unigeschichte/professorenkatalog/

Das Modell alleine kann also nicht ausschlaggebend für die Regeln der Transformation sein. Stattdessen muss insbesondere auf die tatsächlich verwendete Struktur der Daten eingegangen werden.

2.1 Analyse der Modelle

Das Modell wurde mittels dem Tool Vocto[2] als Diagramm visualisiert (siehe Abbildung 3 im Anhang). Mittels der visuellen Hilfe von Vocto entstand auf Basis des alten Modells in kollaborativer Arbeit bereits das neue Modell 2015-16[3] (siehe Abbildung 4 im Anhang). Die grafische Übersicht des Modells ermöglicht auch Benutzern mit wenigen IT Grundkenntnissen ein schnelles und einfaches erfassen der Zusammenhänge.

Die Grafik stellt Klassen (`owl:Class`) in blauen Rechtecken, Properties (`owl:DatatypeProperty` und `owl:FunctionalProperty`) in gelben Ovalen und Relationen (`owl:ObjectProperty`) in grünen Trapezen dar. Die Beziehung werden durch die verbundenen Linien deutlich, wobei Subklassen (`rdfs:subClassOf`) mittels Pfeil auf die Elternklasse zeigen. Im Tool können die Objekte verschoben und weitere Informationen (label, comment etc.) angezeigt werden. In der linken Seitenleiste kann das Modell in Turtle Notation editiert werden, woraufhin die grafische Ansicht direkt angepasst wird.

Grundsätzlich ähneln sich die beiden Modelle. Zentrale Klassen wie `Person`, `Body` und `PeriodOfLife` und deren Beziehungen und Properties sind geblieben. Die auffälligsten Änderungen sind auf der rechten Seite im Bereich des `Documents` zu sehen, die neu Strukturiert wurden. Außerdem wurden die Bezeichnungen der Properties von `QualificationPaper` verkürzt. Die Klasse `Forename` wurde in Properties der Person übertragen und die Klasse `RelationToInstitution`, welche die Beziehung zwischen einer Institution und einem Professor realisierte, als direkte Relation zwischen Körperschaften und Lebensabschnitten von Personen umgesetzt.

Im alten Modell wurden die Familienbeziehungen (`familyParent`, `familyChild` etc.) zwar bereits implementiert, jedoch nicht auf die Daten angewendet. Die dafür verwendeten Properties `fatherName`, `motherName` etc. wurden im neuen Modell entfernt und die Familie im Rahmen der Transformation als neue Relation mit den Eltern als eigene Ressourcen (`Person`) erstellt. Einen Ausschnitt des entsprechenden Mappings zeigt die Abbildung 1.

2.2 Analyse der Daten

Für die Analyse der Daten wurden nach einer Übersicht zunächst verschiedene SQL Anfragen gestellt und in Tabellen festgehalten. Interessant war insbesondere, die verwendeten:

[2] https://github.com/simeonackermann/VocTo
[3] Vocto mit CPM15/16: http://aksw.imn.htwk-leipzig.de:8081/vocto/

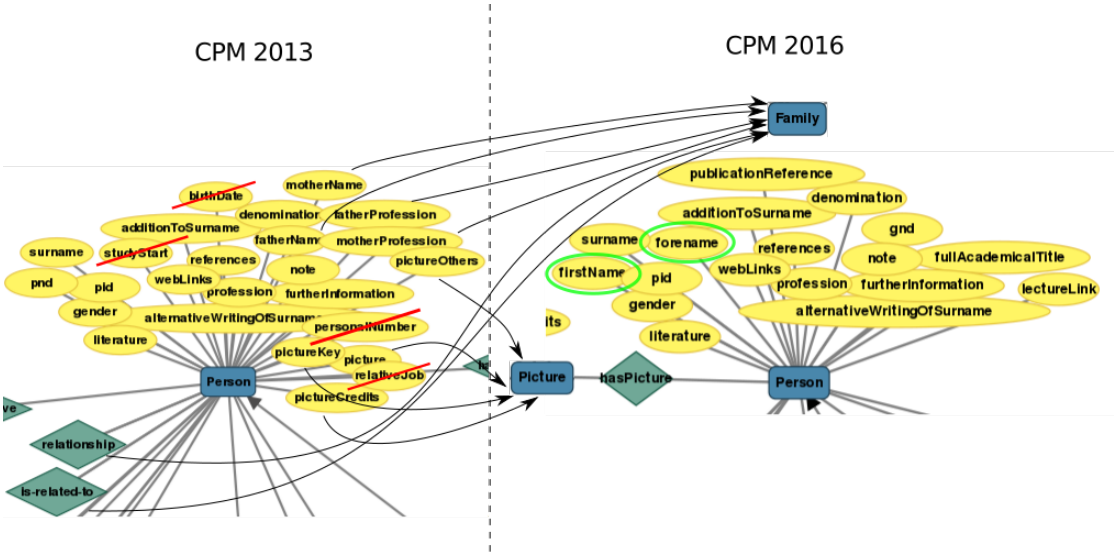

Abbildung 1: Bsp: Mapping der Familien-Properties einer Person

- Klassen-Typen,
- Predikate und
- Predikate je Klassen-Typ (siehe Tabelle 1)

zu zählen und aufzulisten. Durch das Zählen der Vorkommen konnte während der Implementierung verifiziert werden, dass entsprechend die gleiche Anzahl transformiert wird.

Probleme ergaben sich hierbei insbesondere in der teils starken Unterscheidung vom Modell. So wurde zum Beispiel das `birthDate` auch in einer Körperschaft statt der Person verwendet. Außerdem sind nicht eindeutige Relationen (`is-related-to`) zwischen

Tabelle 1: Anzahl der genutzten Predikate je Klassen-Typ (Ausschnitt)

Klassen-Typ	Propertie	Anzahl
cpd:Professor	cpd:additionToSurname	108
	cpd:alternativeWritingOfSurname	491
	cpd:birthCity	1813
	...	
cpd:PeriodOfLife	cpd:from	28
	cpd:functionInOrganisation	9
	cpd:membership	1
	...	

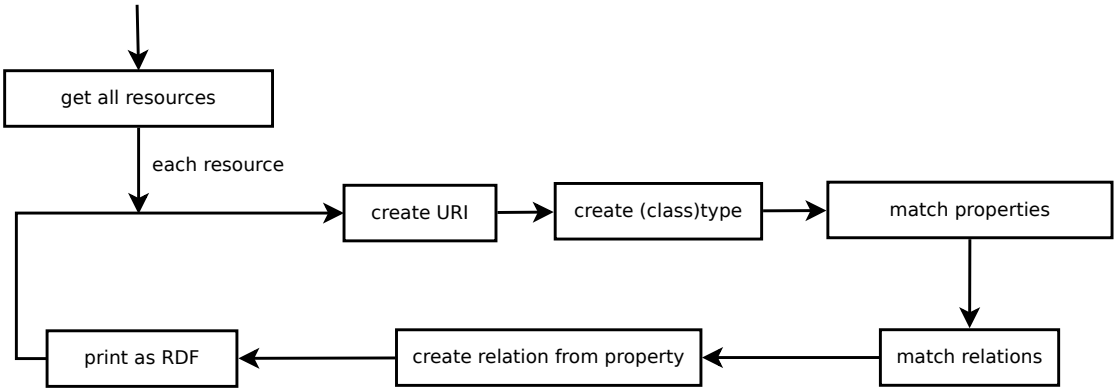

Abbildung 2: Programmablaufplan

Personen vorhanden, die normalerweise mittels der Propertie `relationship` beschrieben wird, jedoch bei mehreren Beziehungen nicht mehr eindeutig ist.

Ein eklatanter Unterschied ist außerdem, dass Ortschaften und Länder im alten Modell als Literal Properties direkt an die Ressourcen gehangen wurden. Dafür werden im neuen Modell eigene neue Ressourcen erstellt, die dann mittels Objekt Propertie[4] verlinkt werden. Die Vorgehensweise dafür wird im letzten Schritt der Implementierung erläutert.

3 Implementierung

Die Implementierung wurde als PHP Anwendung umgesetzt, welche auf die in einer MySQL Datenbank liegenden Daten zugreift und als RDF ausgibt. Der Grundlegende Ablauf ist in Abbildung 2 beschrieben. Mittels einer Start und Offset Angabe wurde in insgesamt 42 Schritten auf alle Ressourcen zugegriffen und für jede Ressource nach Anwendung der fünf Regeln eine neue Ressource erstellt.

Die RDF Ausgabe wurde mittels Raptor[5] auf Korrektheit überprüft und als Tripel in eine Datei geschrieben. Am Ende wurden alle Ergebnisse zusammengeführt und nochmals Duplikate gelöscht, die insbesondere durch die wiederverwendeten Orte entstanden.

Kernfunktion der Transformation ist die Implementierung verschiedener Mapping Templates, welche die Überführung der Ressourcen und Properties aus dem alten Modell in das neue Modell definieren und nachfolgend beschrieben werden.

[4] Im Gegensatz zu Literal Properties, welche als Objekt ein Literal, also z.B. ein String oder Boolean enthalten, verlinken Objekt Properties auf eine andere Ressource. Das Objekt ist damit wiederum eine URI

[5] http://librdf.org/raptor/rapper.html

3.1 URIs

Jeder Ressource muss eine eindeutige URI zugeordnet werden. Für den Professorenkatalog wurden folgende Vorgaben vereinbart:

- Die URI entspricht einer URL, da es sich um einen Webkatalog mit online verfügbaren Ressourcen handelt.
- Die URLs sollen allgemeinen Konventionen für Webadressen folgen, also keine Sonderzeichen oder Leerzeichen enthalten und die maximale Länge von 255 Zeichen nicht überschreiten.
- Die neue Host-Adresse ist: http://catalogus-professorum.org/lipsiensium/ (die alte war http://www.uni-leipzig.de/unigeschichte/professorenkatalog/).
- Die URLs sollen „sprechend“ sein, also bereits durch die Bezeichnung Hinweise auf die Art der Ressource geben.

Für den letzten Punkt der sprechenden URLs wurden zunächst Unterpfade verwendet, um Ressourcen zu beschreiben die einer anderen Ressource untergeordnet sind. So beispielsweise die Geburt einer Person: `cpl:person-max_muster` und `cpl:person-↪ max_muster/birth`. Problematisch ist hierbei jedoch die Turtle Darstellung, welche für jeden neuen Unterpfad ein neues Präfix erzeugt. Also in diesem Fall für die zweite Ressource Birth `cpl_ns0:birth`. Dadurch würden insgesamt unzählige Präfixe entstehen. Stattdessen wurden daher Bindestriche verwendet, also: `cpl:person-max_muster-birth`.

Um die URIs möglichst sprechend zu gestalten, wurde für die meisten Ressourcen der letzte Namensteil der alten URI verwendet und der Typ der Instanz davor geschrieben. Für bisher nicht vorhandene Ressourcen (z.B. die Orte und Familien) musste eine komplett neue URI generiert werden. Dafür wurde jeweils der Wert einer kurzen aber prägnant beschreibenden Eigenschaft der Ressource verwendet. Zum Beispiel für eine neue Stadt deren Name, welcher in den Properties `birthCity`, `deathCity` etc. vergeben wurde.

Da jede URI nur einmal vergeben werden darf um eine Ressource eindeutig zu beschreiben, wurde eine Tabelle der bereits vergebenen URIs erstellt. Vor dem Vergeben einer neuen URI wurde geprüft ob diese bereits vorhanden ist und wenn nötig ein Appendix hinzugefügt. Ausnahmen sind Städte und Körperschaften, die immer die gleiche URI bekommen.

3.2 Class Mapping

Der zweite Schritt besteht aus dem Mapping des alten Klassen-Typs (`rdf:type`) auf den neuen, sowie der Vergabe einer neuen URI. Dafür wird in einem Array für jeden alten Typ die neue Klasse und URI beschrieben. Im Programm wird für jede Ressource der

entsprechende Eintrag aus dem Array auf die neue Ressource angewendet. Im Quelltextbeispiel 1 wird für Instanzen vom Typ `Brother` als neuer Typ `cpd:Person`[6] definiert. Die URI wird mittels einer Callback-Funktion in `urifunc` zurückgegeben, wobei das Präfix `cplPerson`[7] und der letzte Teil der alten URI verwendet wird (`basename($uri)`).

```
"Brother" => array(
  "type" => "cpd:Person",
  "urifunc" => function($uri) { return "cplPerson:" .
      Resource::strToUri(basename($uri)); }
)
```

Quelltext 1: Mapping der Klasse `Brother` nach `Person`, mit Angabe der neuen URI

3.3 Property Mapping

Im dritten Schritt werden je Ressource die neuen Literal Properties definiert. Dies muss für jeden Klassentyp separat beschrieben werden, um den Wert einer Eigenschaft jeweils unterschiedlich festlegen zu können. Im Quellcodebeispiel 2 wird für Ressourcen der Klasse `AcademicSociety` die Propertie `organisationName` als neue Propertie `rdfs:label` und der `organisationType` als `cpd:typeOfBody` festgelegt, da im neuen Modell die Properties entsprechend entfernt bzw. umbenannt wurden.

```
"cpd:AcademicSociety" => array(
  self::$oldBaseUri . "organisationName" =>  "rdfs:label",
  self::$oldBaseUri . "organisationType" =>  "cpd:typeOfBody"
)
```

Quelltext 2: Mapping der Properties in `AcademicSociety`

3.4 Relation Mapping

Im vierten Schritt werden die Relationen (Objekt Propertie) einer Ressource in neue Relationen überführt. Dafür wird wiederum je Klassentyp der Typ der neuen Relation sowie die URI der verlinkten Ressource beschrieben. Im Quelltextbeispiel 3 wird für Ressourcen vom Typ `cpd:Professor` mit der Propertie `has-periods` die neue Relation `cpd:hasPeriod` erstellt. Die URI der verlinkten Ressource im Objekt wird durch eine Callback-Funktion mittels der Hilfsfunktion `newUriFromOrgResource()` geholt, welche für jede alte URI die neue URI wie im ersten Schritt der Transformation zum Erstellen der URIs zurückgibt.

[6] cpd ist Präfix für http://catalogus-professorum.org/cpd/
[7] cplPerson ist Präfix für http://catalogus-professorum.org/lipsiensium/person/

```
"cpd:Professor" => array(
  self::$oldBaseUri . "has-periods" => array(
    "type" => "cpd:hasPeriod",
    "urifunc" => function($olduri) { return
        Resource::newUriFromOrgResource($olduri); }
  )
)
```

Quelltext 3: Mapping der Relation `has-periods` nach `hasPeriods`

3.5 New Relations from Properties Mapping

Die Analyse der Modelle hat gezeigt, dass aus einigen Daten die im alten Datensatz als Properties beschrieben wurden, im neuen Modell als eigene Ressourcen und Relationen erstellt werden. So wurden Orte im alten Datensatz in Properties `locatedAtCity`, `academyCity`, `deathCity` u.a. gespeichert. Mit dem neuen Modell werden Orte als eigene neue Ressource `shv:City`[8] und `shv:Country` beschrieben. Die vorher als Literal Propertie beschriebene Beziehung einer Ressource zu einer Stadt wird also als Relation zu der neu angelegten Orts-Ressource erstellt.

Im letzten Schritt des Mappings werden diese neuen Relationen erstellt. Im Codebeispiel 4 wird für die Properties `birthCity`, `birthDate` etc. einer Ressource vom Typ `cpd:Professor` die neue Relation `cps:hasPeriod` erstellt. Die neue Ressource der Relation ist vom Typ `cpd:Birth` und erhält die Liste der Properties. Die beschriebenen Mappings der Transformation werden nun rekursiv auf die Geburt angewendet, um URI, Klassentyp, Properties etc. zu erstellen. Die neue URI der Geburt wird als Wert des Geburt-Lebensabschnittes für den Professor verwendet. Im Codebeispiel 5 wird die Transformation anhand eines Minimalbeispiels verdeutlicht. Nach der Transformation wurden die Ressourcen `Max-Mustermann-Birth` und `Leipzig` erstellt und die URI jeweils als Wert der Objekt Propertie hinzugefügt.

```
"cpd:Professor" => array(
  "cpd:Birth" => array(
    "type" => "cpd:hasPeriod"
    "properties" => array(
      self::$oldBaseUri . "birthCity",
      self::$oldBaseUri . "birthDate",
      self::$oldBaseUri . "birthState",
      self::$oldBaseUri . "birthLand"
    )
  )
)
```

Quelltext 4: Mapping für neue Relationen aus Properties

[8] shv ist Präfix für http://ns.aksw.org/spatialHierarchy/ und beschreibt ortsbezogene Daten

```
# alt
<Max-Mustermann> a cpd:Professor ;
  cpd:birthCity "Leipzig" .

# neu, nach der Transformation
<Max-Mustermann> a cpd:Professor ;
  cpd:hasPeriod <Max-Mustermann-Birth> .

<Max-Mustermann-Birth> a cpd:Birth ;
  cpd:periodPlace <Leipzig> .

<Leipzig> a shv:City ;
  rdfs:label "Leipzig" .
```

QUELLTEXT 5: Überführung der Geburtsstadt in eine neue Ressource und Relation (in Turtle Notation)

4 Auswertung

Aufbauend auf die Resultate der vorausgegangenen Analyse wurden Mapping-Templates von Klassentypen, Properties und Relationen erstellt und implementiert. Die Transformation des Datensatzes in das neue Modell konnte dadurch durchgeführt und verifiziert werden. Die entstandene PHP Anwendung ist online auf GitHub verfügbar[9].

Aus ursprünglich 215.975 Tripeln und 21.493 Ressourcen sind nach der Transformation insgesamt 268.393 Tripel 41.418 Ressourcen entstanden, da Städte, Länder und verschiedene Lebensabschnitte als neue Ressourcen erstellt wurden. Die Normalisierung dieser Ressourcen wird als erhebliche Verbesserung des Professorenkatalogs betrachtet.

Die syntaktische Korrektheit der RDF Daten wurde mittels Raptor getestet und bestätigt. Erste manuelle Untersuchungen mittels OntoWiki konnten die korrekte Verwendung der Klassen und Properties des neuen Modells bestätigen. Weitere Untersuchungen zur semantischen Korrektheit stehen noch aus. So wurden im alten Datensatz beispielsweise Städte in den Properties unterschiedlich oder mit Rechtschreibfehlern geschrieben. Bei der Transformation sind dadurch mehrere Ressourcen des eigentlich gleichen Objektes entstanden (z.B. `cpd:Leipzig` und `cpd:Leizpig`). Die Korrektur dieser Fehler ist händisch vorzunehmen.

In Einzelfällen reichte das implementierte Mapping nicht aus. So wurde der individuelle Fall zum Erstellen von Familien teils fest in das Programm geschrieben. Problematisch hierbei war, dass die neue Ressource `cpd:Family` Backlinks wie `familyFather` oder `familyChild` zur ursprünglichen Ressource enthält sowie die Eltern (bzw. Kinder) auch

[9] https://github.com/simeonackermann/cpl_transformation

jeweils als neue Ressourcen angelegt werden muss. Eine Abstraktion dafür war im Rahmen dieser Ausarbeitung zeitlich zu aufwändig, ist jedoch prinzipiell möglich.

Nicht Teil dieser Arbeit war die Transformation einer Änderungshistory für die Ressourcen. Diese enthält für jede Ressource Informationen zur zeitlichen Veränderung durch verschiedene Benutzer. Dies sollte jedoch ohne großen Aufwand möglich sein, da lediglich die URIs der Statements in der History ersetzt werden müssten.

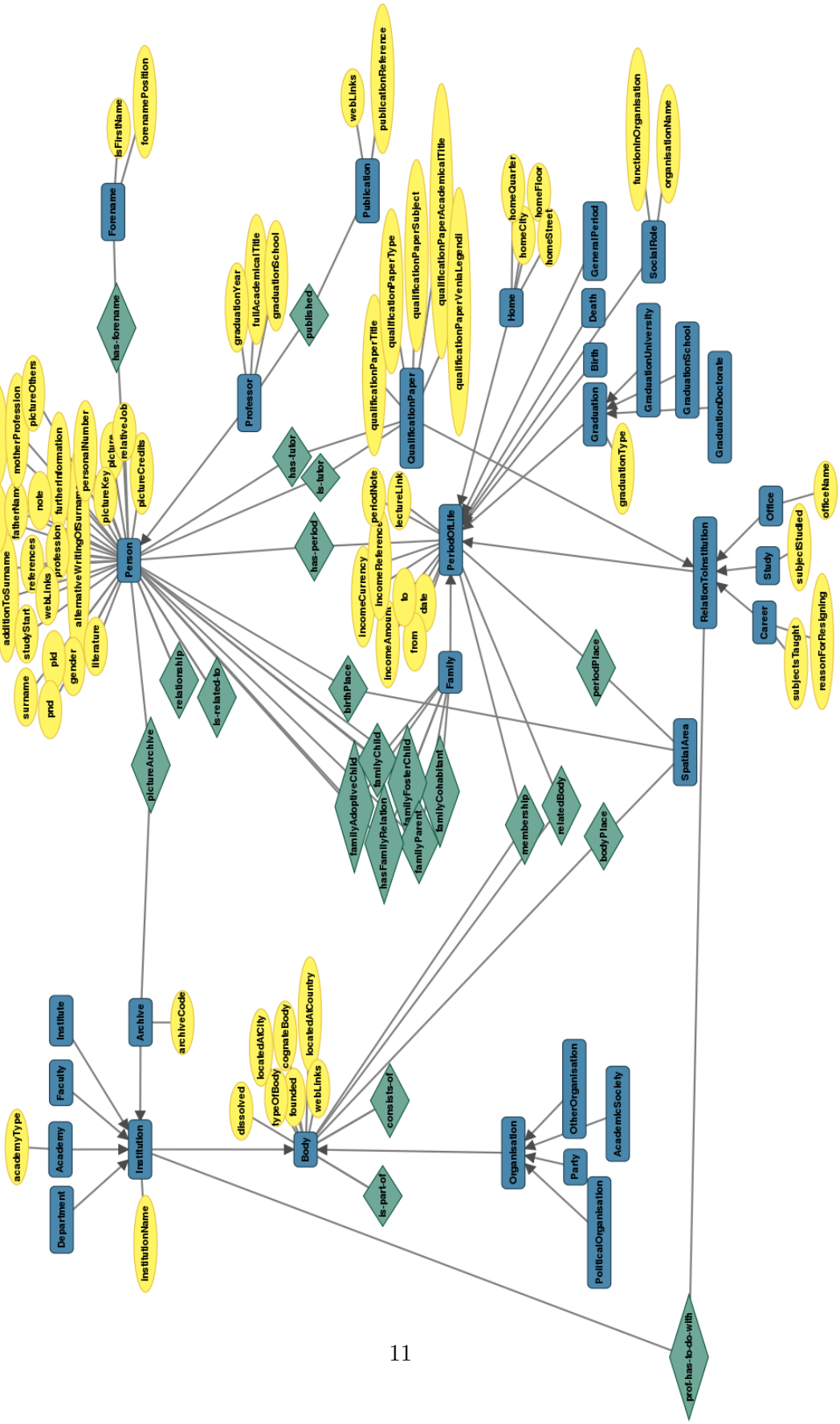

Abbildung 3: CPM Modell 13/14

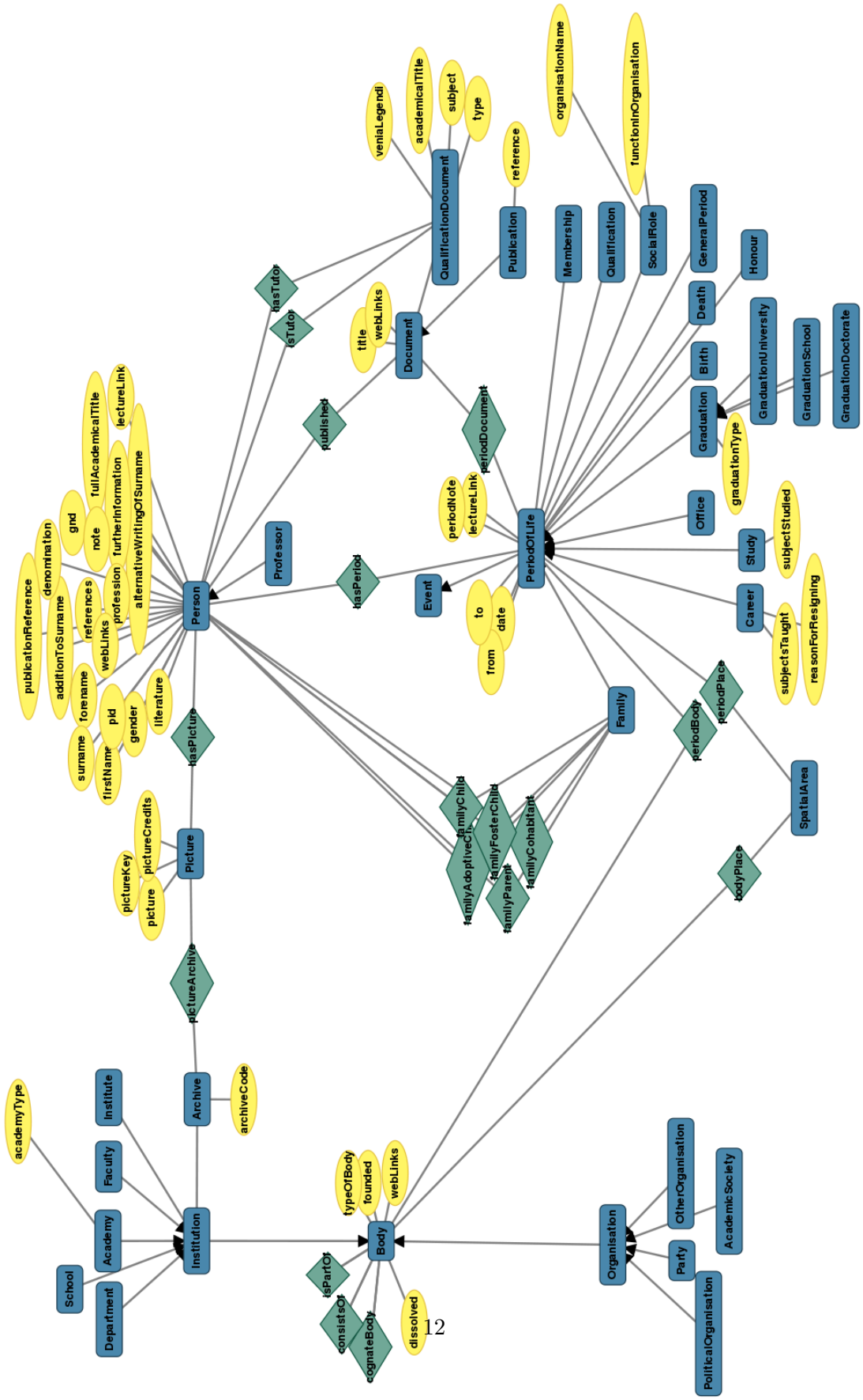

Abbildung 4: CPM Modell 15/16