M2 – DAC - LODAS - 2019

MU5IN860 Linked Open Data, Apprentisssage Symbolique

Bernd Amann

SU

15 octobre 2019

Intégration de Données avec RDF

1 Linked-Open-Data Cloud

2 LOD : Construction

3 LOD : Interrogation

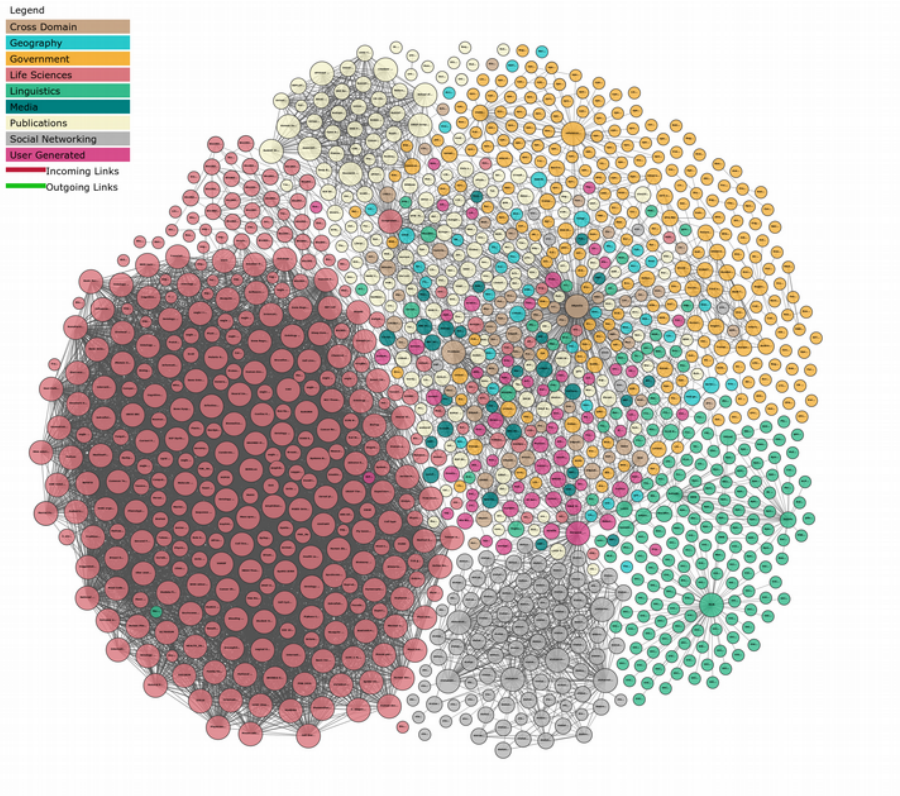
Outline

1 Linked-Open-Data Cloud

2 LOD : Construction

3 LOD : Interrogation

LOD : Linked-Open-Data Cloud



Linked Open Data Cloud

- `http://lod-cloud.net/`
- graphe de 1 224 datasets RDF reliés par 16 113 liens (Juin 2018)
- chaque dataset RDF évolue indépendamment et est lié à d'autres dataset RDF
- rafraîchir les données régulièrement est coûteux → architectures fédérées

"Linking Open Data cloud diagram 2017, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. http ://lod-cloud.net/"

Datasets LOD

Dataset LOD

Collection de datasets RDF reliées et accessibles à travers un seul point d'accès.

Objectif et tâches

Construire, relier et interroger des datasets RDF :

- Collecte et transformation de données web en datasets RDF ;
- Publication sous forme de fichiers ou SPARQL endpoints ;
- Intégration de datasets RDF ;
- Interrogation structurée avec SPARQL.

Exemples de datasets LOD

- YAGO : `http://yago-knowledge.org/`
- DBPedia : `http://dbpedia.org/`
- Wikidata : `http://wikidata.org/`
- Freebase : `http://freebase.com/` (Google Knowledge Graph)
- OpenCyc : `http://opencyc.org/` (Cycorp)

Linked-Open-Data : Objectifs et Tâches

Objectif

Construire un *dataset LOD cohérent* sur des datasets RDF autonomes et hétérogènes.

Tâches

- représenter des données hétérogènes, complexes avec et sans schémas
- définir des vues cohérentes
- résoudre des conflits syntaxiques et sémantiques
- interroger les données intégrées

Linked-Open-Data : Objectifs et Tâches

Objectif

Construire un *dataset LOD cohérent* sur des datasets RDF autonomes et hétérogènes.

Tâches

- représenter des données hétérogènes, complexes avec et sans schémas $\rightarrow$ graphes RDF
- définir des vues cohérentes
- résoudre des conflits syntaxiques et sémantiques
- interroger les données intégrées

Linked-Open-Data : Objectifs et Tâches

Objectif

Construire un *dataset LOD cohérent* sur des datasets RDF autonomes et hétérogènes.

Tâches

- représenter des données hétérogènes, complexes avec et sans schémas $\rightarrow$ graphes RDF
- définir des vues cohérentes $\rightarrow$ RDFS / OWL
- résoudre des conflits syntaxiques et sémantiques
- interroger les données intégrées

Linked-Open-Data : Objectifs et Tâches

Objectif

Construire un *dataset LOD cohérent* sur des datasets RDF autonomes et hétérogènes.

Tâches

- représenter des données hétérogènes, complexes avec et sans schémas → graphes RDF
- définir des vues cohérentes → RDFS / OWL
- résoudre des conflits syntaxiques et sémantiques → règles/entailment, SPARQL
- interroger les données intégrées

Linked-Open-Data : Objectifs et Tâches

Objectif

Construire un *dataset LOD cohérent* sur des datasets RDF autonomes et hétérogènes.

Tâches

- représenter des données hétérogènes, complexes avec et sans schémas → graphes RDF
- définir des vues cohérentes → RDFS / OWL
- résoudre des conflits syntaxiques et sémantiques → règles/entailment, SPARQL
- interroger les données intégrées → SPARQL

LOD : Extraction d'informations

Objectif

Transformer des ressources web en datasets RDF.

Ressources Web

- sites web : tableaux HTML, liens, méta-données
- taxonomies et ontologies existantes (Wordnet, catégories Wikipedia)
- fichiers : CSV, XML, JSON
- autres datasets RDF

Tâches

- Wrapper induction : construction automatique de wrappers à partir d'exemples ;
- Link prediction : prédiction de liens dans un graphe incomplet ;

Application de techniques d'apprentissage supervisé ; similaire aux tâches de recommandation

LOD : Intégration de datasets RDF

Objectif

Détecter et résoudre des conflits sémantiques au niveau des schémas et des données RDF.

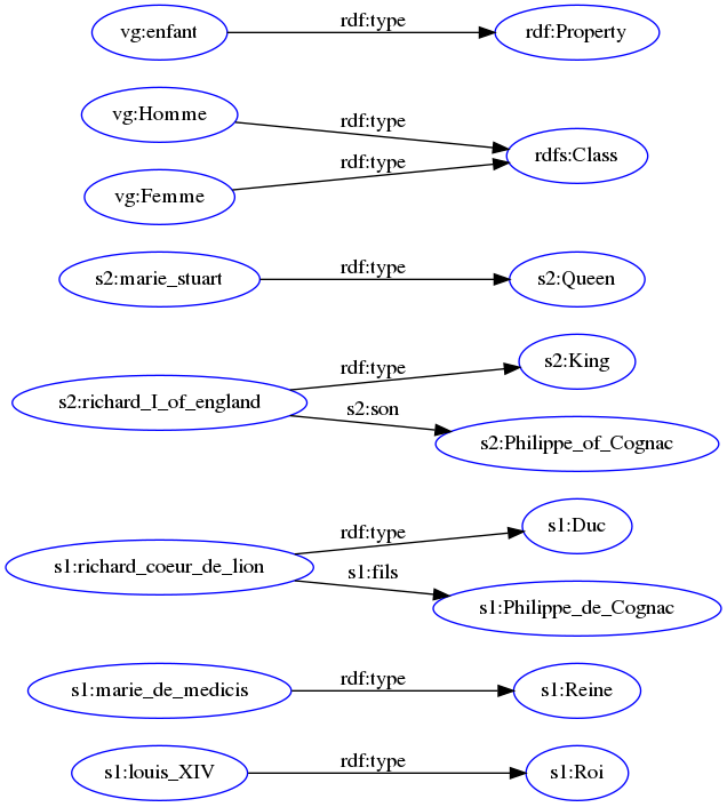
Tâches

- Alignement de schémas : extraire des relations sémantiques entre des schémas RDFS/OWL ;
- Résolution d'entités : extraction de contraintes sémantiques entre les ressources RDF.

Exemple

Données provenant de trois sources

```
s1:louis_XIV a s1:Roi .
s1:marie_de_medicis a s1:Reine .
s1:richard_coeur_de_lion a s1:Duc;
      s1:fils s1:Philippe_de_Cognac
s2:richard_I_of_england a s2:King;
      s2:son s2:Philippe_of_Cognac .
s2:marie_stuart a s2:Queen .
```



Vue intégrée

```
vg:Homme a rdfs:Class .
vg:Femme a rdfs:Class .
vg:enfant a rdf:Property .
```

Exemple

Alignement de schémas : rdfs:subClassOf

```
s1:Roi owl:equivalentClass s2:King .
s1:Reine owl:equivalentClass s2:Queen .
s2:King owl:disjointWith s1:Duc .
s1:fils owl:equivalentProperty s2:son .

s1:Roi rdfs:subClassOf vg:Homme .
s1:Reine rdfs:subClassOf vg:Femme .
s1:fils owl:subPropertyOf vg:enfant .
```

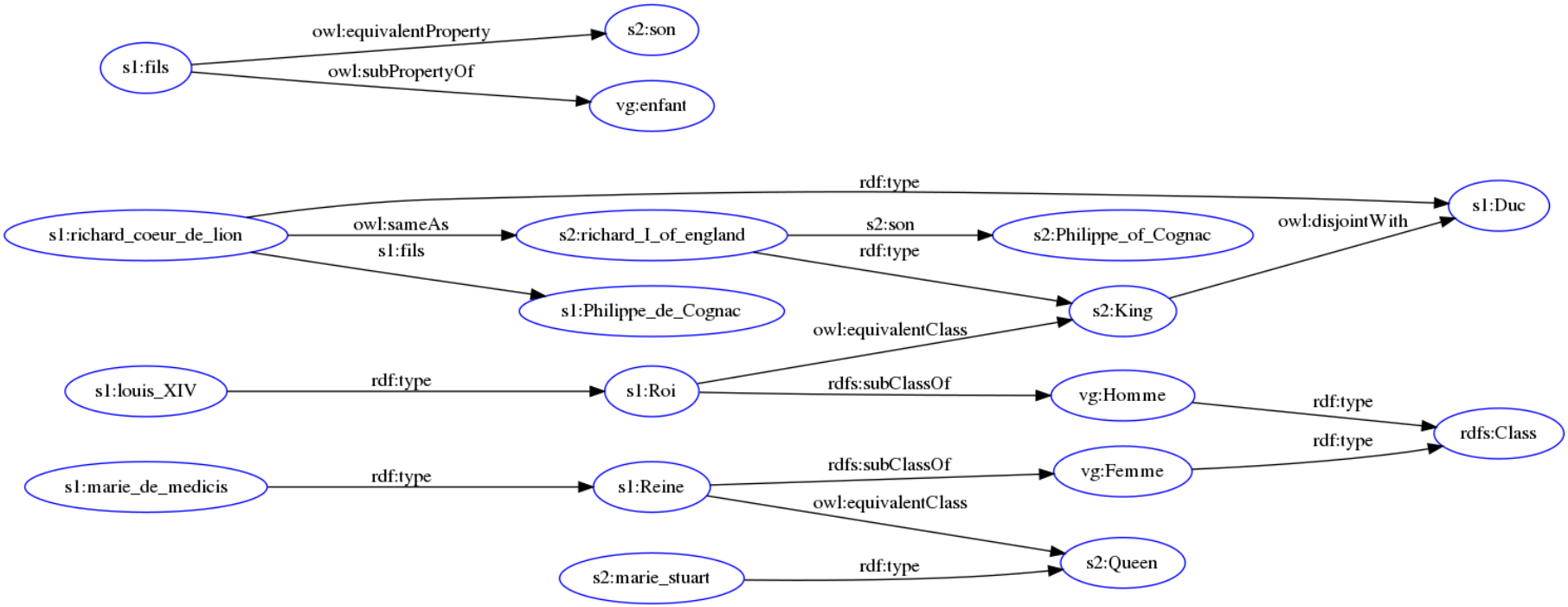
Résolution d'entités : owl:sameAs

```
s1:richard_coeur_de_lion owl:sameAs s2:richard_I_of_england .
s2:Philippe_of_Cognac owl:sameAs s1:Philippe_de_Cognac .
```

Problème

Découverte automatique d'alignements rdfs:subClassOf et mapping owl:sameAs.

Exemple



- owl:equivalentClass et owl:equivalentProperty peuvent être défini avec rdfs :subClassOf et rdfs :subPropertyOf.
- On peut peut inférer s1:King rdfs :subClassOf vg:Homme . et s1:Queen rdfs:subClassOf vg:Femme .
- Où se trouve la contradiction ?

Outline

1 Linked-Open-Data Cloud

2 LOD : Construction

3 LOD : Interrogation

2 - LOD : Construction

- Alignement de schémas
- Exemple : Coma 3.0
- Résolution d'entités

Alignement de schémas (schema matching)

Relations de matching

Une relation de matching $m = (id, e, e', R, c)$ est un 5-tuple :

- identifiant id
- deux éléments e, e' : classes, propriétés, chemins, sous-graphes, ...
- relation R :
  - ensembliste / sémantique : équivalence, inclusion, séparation, ...
  - fonction : age=now()-birthdate, fullname=firstname+name, ...
  - booléen : birthDate < deathDate, father $\neq$ mother
- confiance $c \in [0, 1]$

Chaque matching peut être représenté sous forme d'un triplet RDF.

Alignement

Un alignement entre deux schémas S et T est un ensemble de matching (id, e, e', R, c) entre des éléments $e \in S$ et $e' \in T$.

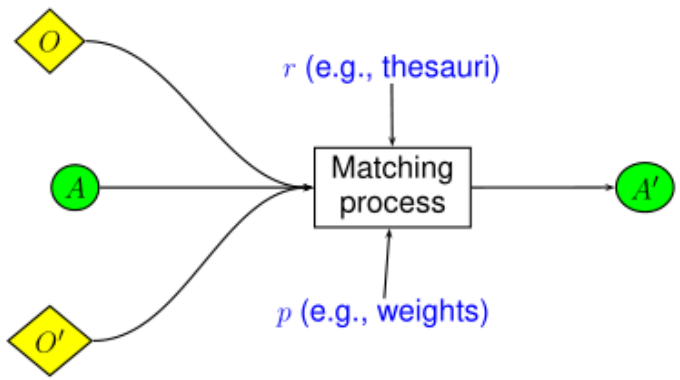
Détection d'alignement (matching)

Objectif

Découverte automatique d'alignements entre de deux schémas.

Entrées

- deux schémas O et O'
- un alignement A (potentiellement vide)
- des paramètres : thésaurus, poids, ...



Sortie

- un nouvel alignement A'

Techniques de schema matching

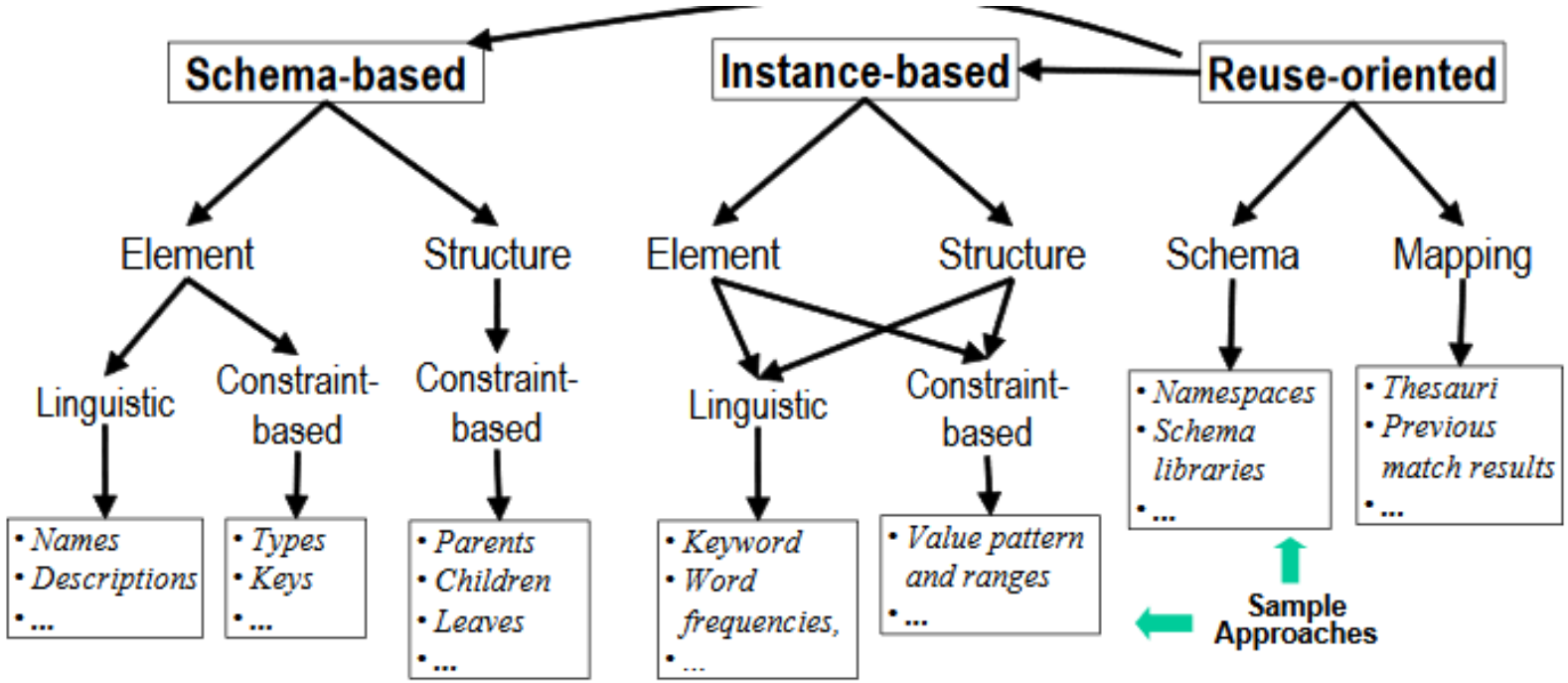
Matching élémentaire

- noms (classes, propriétés) :
  - similarité chaînes de caractères (Levenshtein, ngram, ...)
  - équivalences sémantiques (Wordnet)
  - Techniques NLP
- contraintes (valeurs) : type (date, entier), domaine (intervalle)

Matching structurel

- schéma/ontologie = graphe étiqueté de concepts/classes reliés par des propriétés
- utilisation des matching élémentaires
- heuristiques structurelles : similarité de chemins, partage d'ancêtres/descendants

Classification des méthodes d'alignement



Further criteria:

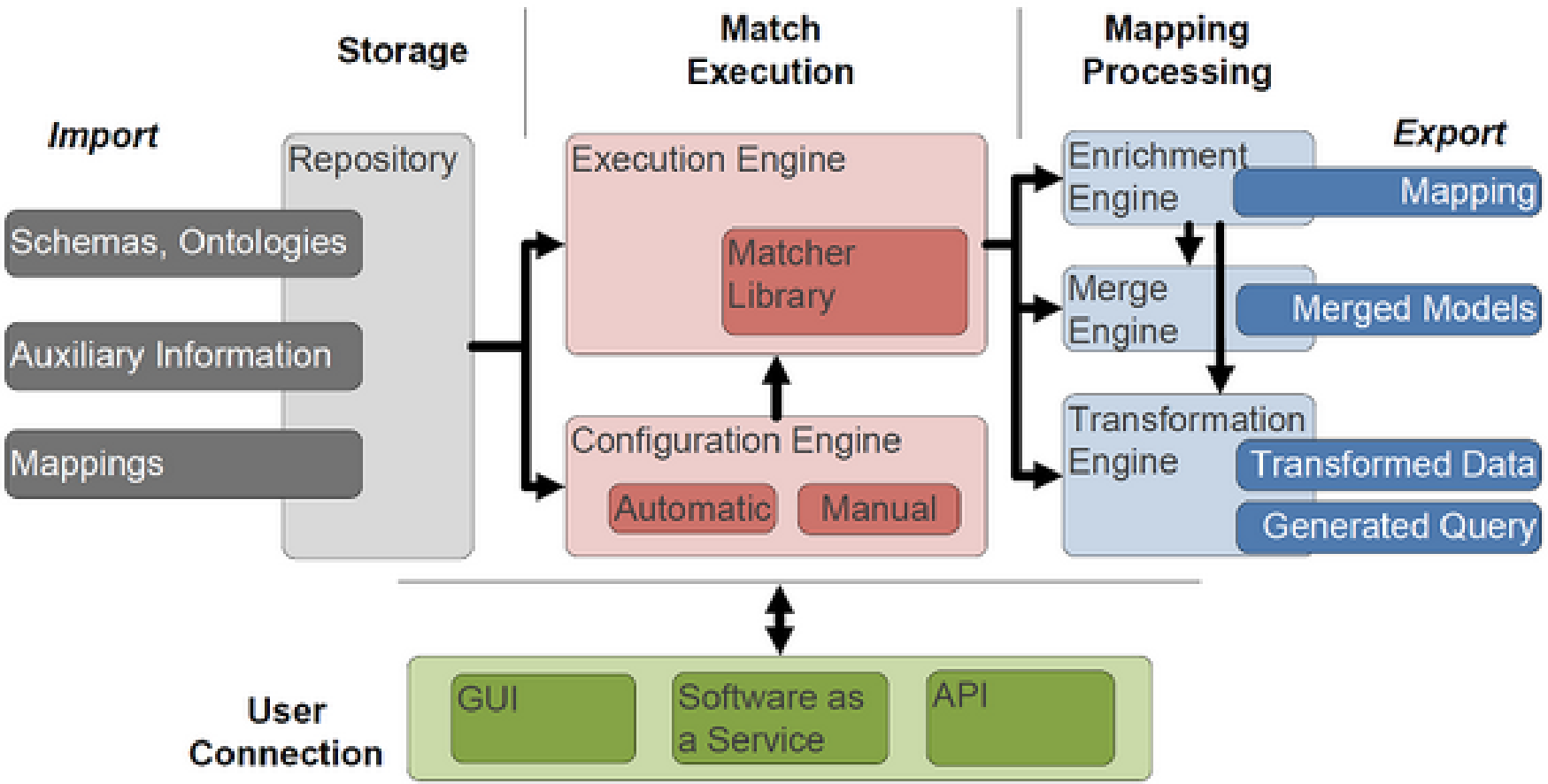
•Hybrid vs. composite combination •Match cardinality

http://www.mkbergman.com/1769/50-ontology-mapping-and-alignment-tools/

2 - LOD : Construction

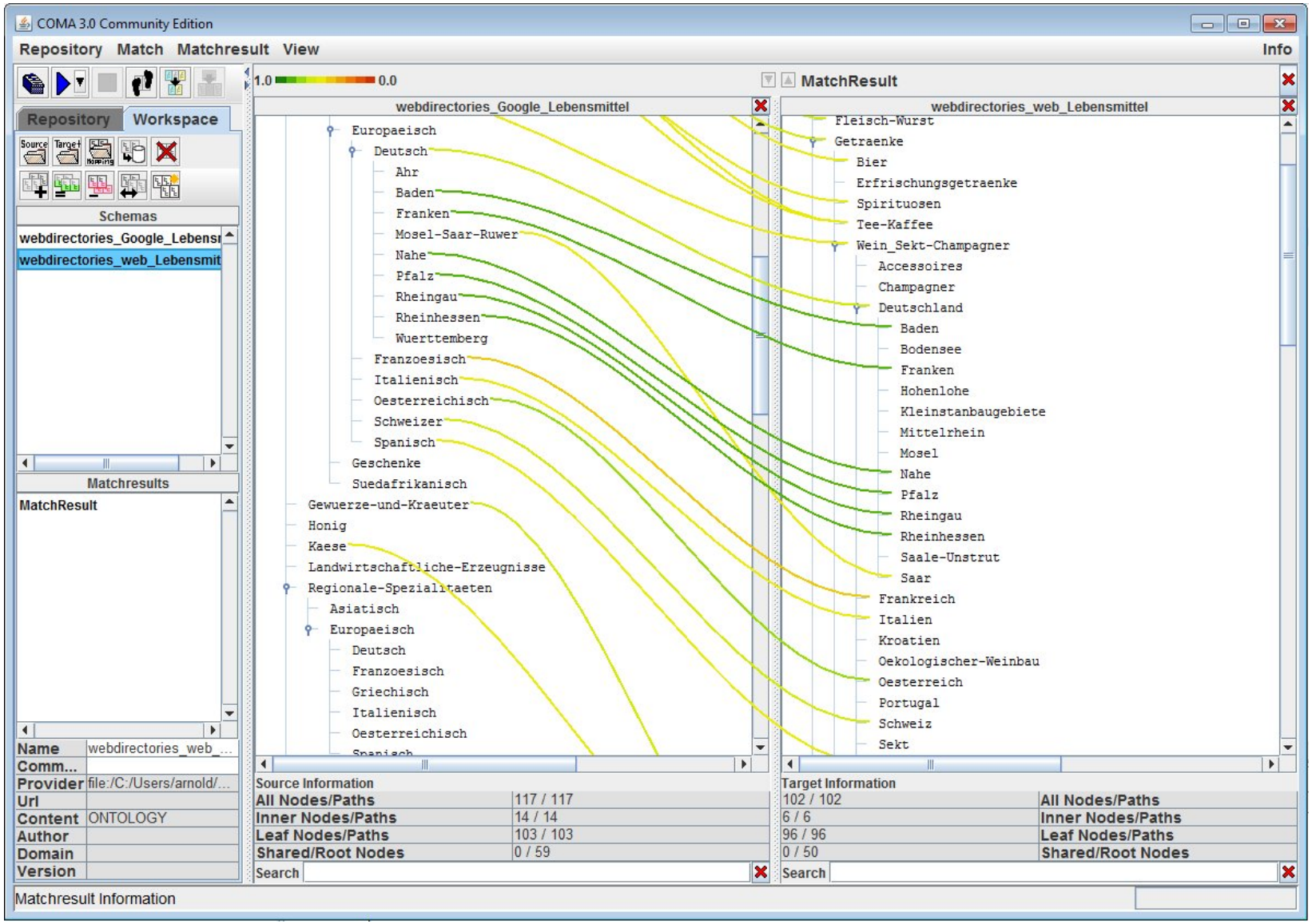
- Alignement de schémas
- Exemple : Coma 3.0
- Résolution d'entités

Coma 3.0 : Architecture

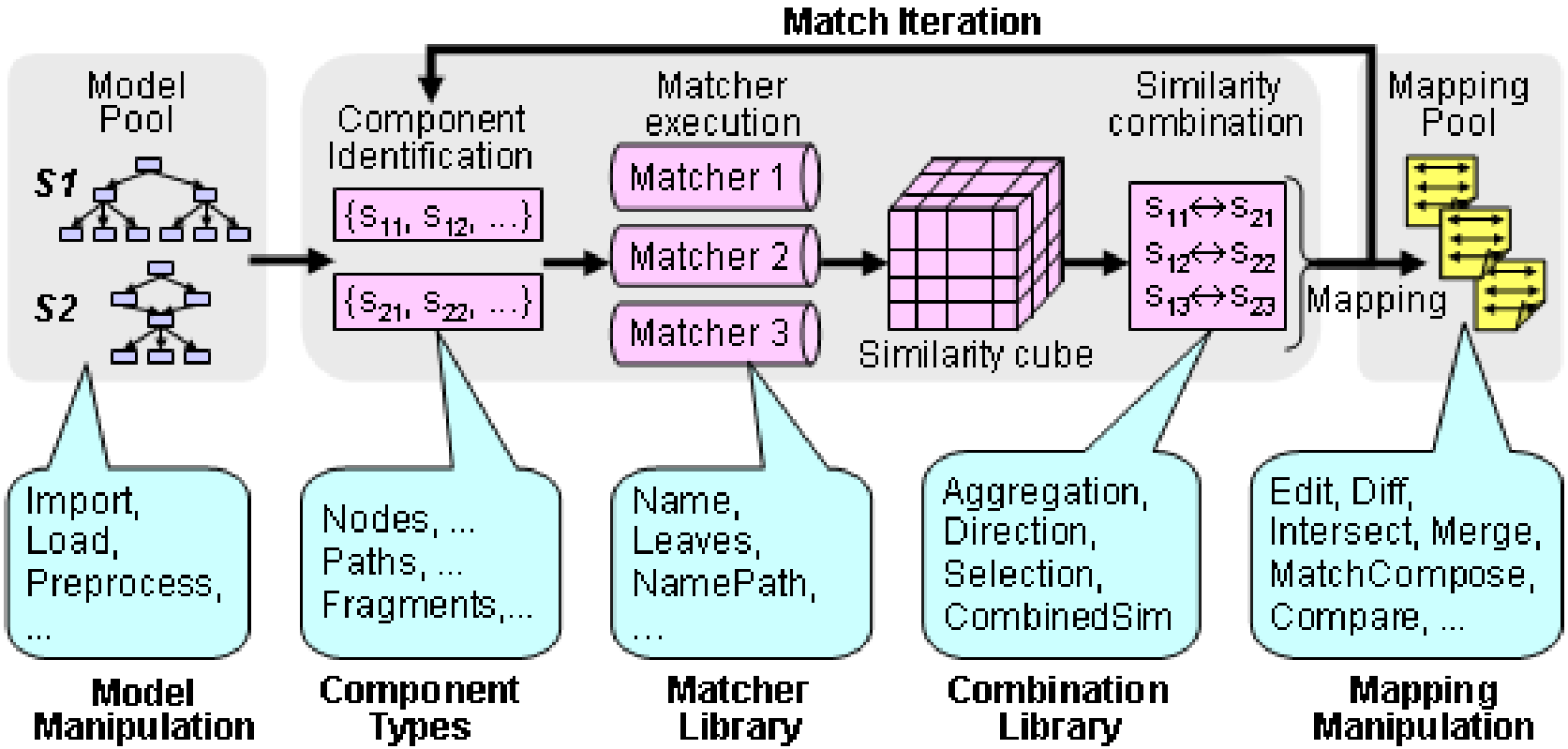


https://dbs.uni-leipzig.de/de/Research/coma.html

Coma 3.0 : Interface



Coma 3.0 : Processus de matching



Coma 3.0 : Processus de matching

- éléments : noeuds, chemins, sous-graphes
- element identification : transformation de schémas S_1 et S_2 en ensemble d'éléments
- 15 matchers séparés en 2 classes (hybrid, combined) : matchings sur les noms, types et structure
- similarity cube : matrice de similarité de dimension $k * m * n$: k matchers, m éléments de S_1, n éléments de S_2) ;
- similarity combinaison : agrégation et classement des k valeurs de similarité pour chaque couple d'éléments ;

Coma 3.0 : Processus de matching

1. *Element identification:* Paths

S_1 **Paths** {*ShipTo.shipToCity, ShipTo.shipToStreet, ... }*
S_2 **Paths** {*DeliverTo.Address.City, ...}*

2. *Matcher execution:* Name, NamePath

Similarity Cube

S_1 Element	S_2 Element	Matcher	Sim
ShipTo.shipToCity	*DeliverTo.Address.City*	Name	0.6
		NamePath	0.8
ShipTo.shipToStreet	*DeliverTo.Address.City*	Name	0.5
		NamePath	0.7

3. *Aggregation:* Average

Similarity Matrix

S_1 Element	S_2 Element	Sim
ShipTo.shipToCity	*DeliverTo.Address.City*	0.7
ShipTo.shipToStreet	*DeliverTo.Address.City*	0.6

4. *Direction:* SmallLarge

Rank S1 elements for each element of larger schema S2

S_1 Element	S_2 Element	Sim
ShipTo.shipToCity	*DeliverTo.Address.City*	0.7
ShipTo.shipToStreet	*DeliverTo.Address.City*	0.6

5. *Selection:* Max1

Match Result

S_1 Element	S_2 Element	Sim
ShipTo.shipToCity	*DeliverTo.Address.City*	0.7

2 - LOD : Construction

- Alignement de schémas
- Exemple : Coma 3.0
- Résolution d'entités

Alignement de schémas versus mapping d'instances

Mapping d'instances

Un mapping d'instances définit des *relations sémantiques* (équivalence, part-of,...) entre des *ressources RDF*.

Alignement de schémas et mapping d'instances

- L'alignement de schémas est une étape préliminaire pour générer des mappings d'instances.
- Les alignements de schémas sont utilisés pour inférer des nouveaux triplets RDF (extensions de datasets).
- Les mappings d'instances permettent d'identifier des ressources qui correspondent à une même entité (résolution d'entités).

Contexte du LOD

- Alignement : trouver des triplets owl:equivalentClass et owl:equivalentProperty.
- Mapping : génération de triples owl:sameAs (résolution d'entités)

Ressources RDF et Entités

Graphes RDF

- ressources R, propriétés P et de littéraux L
- graphe : ensemble de triplets $T \subseteq R \times P \times (R \cup L)$

Dans la suite on utilise également la notation $p(r, s)$ pour désigner un triplet r p s..

Interprétation et description

- Ensemble d'entités E
- $I : R \rightarrow E$: chaque ressource $r \in R$ *décrit* une entité $I(r) \in E$ du monde réel
- $D : R \rightarrow 2^T$ associe à chaque ressource r l'ensemble de triplets r p x.
- L'ensemble de triplets $D(r)$ est une description de l'entité $I(r)$
- Plusieurs ressources r_i peuvent décrire la même entité.

Le motif de triplet r ?p ?o. retourne la description d'une ressource r

Résolution d'entités

Objectif

Identifier (automatiquement) des ressources RDF qui décrivent les mêmes entités à partir de leurs descriptions.

Résolution d'entités (entity matching)

- définir des fonctions de *matching* $M : R \times R \rightarrow \{true, false\}$ tel que $I(r_1) = I(r_2)$ ssi $M(r_1, r_2)$

La relation M correspond à la relation owl:sameAs.

- Le calcul de M utilise des fonctions de *similarité* sim_i appliquées aux *descriptions* et des *bornes* b_i :
  - $M(r_1, r_2) = true$ si $sim_i(D(r_1), D(r_2)) \geq b_i$
  - $M(r_1, r_2) = false$ sinon

Fonctions de similarité

Similarité ensembliste

- proportion de valeurs de propriétés partagées
- exemple : distance Jaccard

Similarité par valeur

- similarité des valeurs d'une propriété partagée
- exemples : Levenshtein, Jaro Winkler, Dice, ...

Similarité contextuelle

- les fonctions de similarité s'appliquent aussi aux noms de propriétés et aux noms des classes
- utilisation récursive du mapping M.

Similarité contextuelle : Exemple *lindaSim*

Contexte $C(r)$ d'une ressource r

$C(r) = D(r) \times W$ où W est un vecteur qui associe à chaque triplet dans $D(r)$ un poids.

Similarité contextuelle

$$contextSim(r_i, r_j) = \sum_{(r_i,p_i,o_i,w_i) \in C(r_i)} max_{(r_j,p_j,o_j,w_j) \in C(r_j)}(w_i \times sim(p_i, p_j) \times M(o_i, o_j))$$

où $|C(r_i)| \leq |C(r_j)|$, $x_{i,j} = 1$ et $sim(p_i, p_j)$ est la similarité Levenshtein entre les deux propriétés p_i et p_j.

$contextSim(r_i, r_j)$ dépend du nombre d'entités similaires (o_i, o_j) qui sont reliées par des propriétés similaires (p_i, p_j) avec r_i et r_j.

Similarité globale (Linda)

$$lindaSim(r_i, r_j) = contentSim(r_i, r_j) + \beta \times contextSim(C(r_i), C(r_j)) - \theta$$

où β contrôle l'influence du contexte et θ normalise les valeurs autour de 0 (pour classer les pairs de descriptions par leur évidence).

Propriétés du mapping M

Propriétés de M

- commutativité : $M(r_1, r_2) \rightarrow M(r_2, r_1)$
- transitivité : $M(r_1, r_2), M(r_2, r_3) \rightarrow M(r_1, r_3)$
- dépendance de doublons : si $p(r_1, o_1)$ et $p(r_2, o_2)$ et $M(r_1, r_2)$, alors $M(o_1, o_2)$ devient plus probable
- dépendance de fusion : si $M(r_1, r_2)$ alors on peut générer une nouvelle ressource r_3 avec la description $D(r_3) = \{p(r_3, o) | p(r_1, o) \in D(r_2) oup(r_2, o) \in D(r_1)\}$ qui peut être utilisée dans le processus.

$p(r, o)$ correspond au triplet r p o .

Entité

M génère un *partitionnement* P de R : $P(r_i) = \{r_i\} \cup \{r_j | r_j \in R : M(r_i, r_j)\}$. Chaque partition représente une entité.

Hypothèse

L'application itérative de la fonction M augmente le produit *rappel* $*$ *precision*.

Algorithme de résolution d'entités

Algorithme *matchEntity*

Data: liste de ressources R (descriptions) ; fonctions de matching M
Result: ensemble de partitions P
$P = \emptyset$;
$R_{new} = R$;
while $R_{new} \neq \emptyset$ **do**
  $R_{tmp} = \emptyset$;
  for $r_x \in R$ **do**
    $R_{new} = R_{new} - \{r_x\}$;
    for $r_y \in R_{new}$ **do**
      if $M(r_x, r_y)$ **then**
        $P(r_x) = P(r_x) \cup \{r_y\}$;
        create r_n where $D(r_n) = D(r_x) \cup D(r_y)$ (fusion) $R_{tmp} = R_{tmp} \cup \{r_n\} - \{r_x, r_y\}$;
      end
    end
  end
  $R_{new} = R_{tmp}$;
end

Complexité

- $C = k \times |R|^2/2 \times cout(M)$
- k dépend du nombre de fusions (ressources générées) et M peut être très coûteux !

Méthode de blocage

Objectif

Réduire le nombre de comparaisons en partitionnant des descriptions dans des blocs

Algorithme

1. effectuer un *partitionnement approximatif* de R en b *blocs* R_i.
2. appliquer l'algorithme *matchEntity* à chaque bloc R_i.

Complexité

- $C = k \times b \times |R_i|^2/2 \times cout(M)$
- on suppose que $b \times |R_i|^2/2 \ll |R|^2/2$

Problème

- Définir une méthode de partitionnement
  - efficace : linaire dans le nombre de descriptions (la taille de R)
  - précise : si r_1 et r_2 sont dans la même partition, alors $M(r_1, r_2)$
  - correct : si $M(r_1, r_2)$ alors r_1 et r_2 se trouvent dans la même partition

Méthodes de blocage

Clés de blocage

- Clés de blocage : le bloc d'une description est déterminé par une fonction de similarité entre la description et la valeur de la clé de blocage.
- Utilisation de plusieurs clés : la même description peut se retrouver dans plusieurs blocs.

Méthodes de blocage

Il existe de nombreuses méthodes qui se distinguent par rapport aux clés de blocage utilisés :

- méthode standard :
  - clé = sous-ensemble de propriétés ; regroupement par clés
- voisinage triés (sorted neighborhood) :
  - clé = compression de $D(r)$; regroupement par sous-séquences ordonnées
- clustering « canopée » :
  - « k-means » avec fonction de similarité
- attribute cluster blocking :
  - identifier et regrouper les attributs similaires (par valeur) partagés entre plusieurs descriptions

Outline

1 Linked-Open-Data Cloud

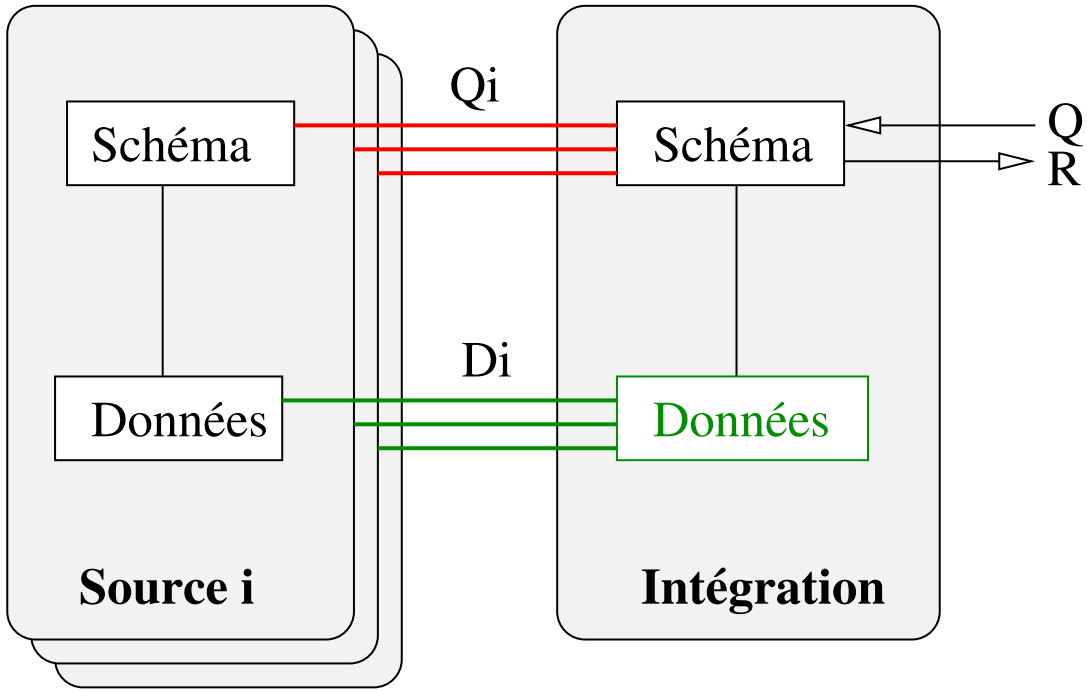
2 LOD : Construction

3 LOD : Interrogation

3 - LOD : Interrogation

- Architectures d'intégration
- Systèmes RDF fédérés

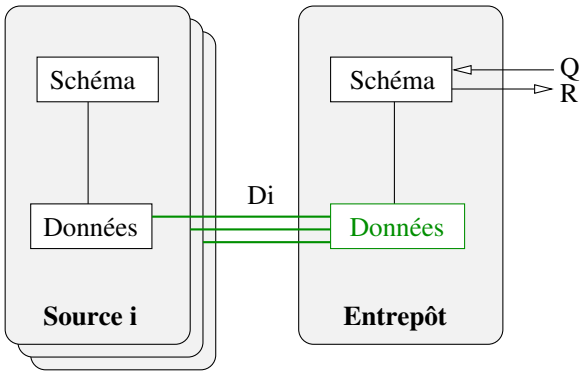
Migrer les requêtes ou les données ?



Deux architectures

1. Médiation de requêtes
2. Entrepôt de données

Migration de données : Entrepôt de données



Principe

- migration des données D_i vers un entrepôt
- processus ETL (Extraction/Transformation/Chargement)
- interrogation de l'entrepôt

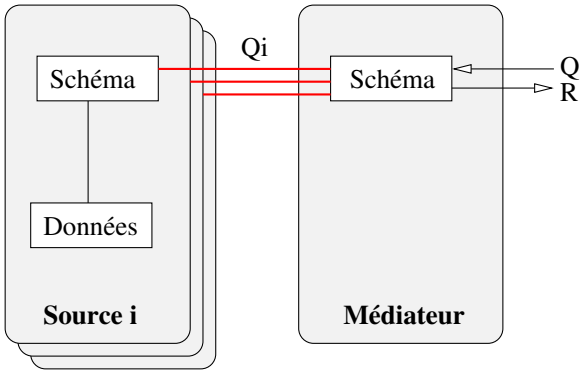
Avantages

- temps de réponse aux requêtes
- possibilité
  - de personnaliser/annoter les données
  - d'archiver et gérer des versions
  - d'interroger de ressources *passives* (fichiers, documents)

Inconvénients / problèmes

- rafraîchissement de l'entrepôt
- stockage et gestion de grands volumes de données

Migration de requêtes : Médiateurs



Principe

- réécriture de la requête Q en requêtes sources Q_i
- évaluation des sous-requêtes par les sources
- intégration des résultats par le médiateur

Avantages

- fraîcheur des données interrogées
- espace disque indépendant de la taille des données

Inconvénients

- coûts de communication, performance du réseau
- réécriture des requêtes
- dépendance des interfaces/capacités des sources

Deux Approches de Médiation : LAV et GAV

Problème de médiation

- Réécriture d'une requête Q sur un médiateur M en une requête équivalent Q' sur des sources $S_1, ...S_n$.

Approche « Global as View »

- Les données du médiateurs M (tables, classes, propriétés) sont définis comme des vues sur les sources S_i.
- La requête Q est réécrite en requête équivalente remplaçant les vues par leur définition (unfolding).

Approche « Local as View »

- Les sources S_i sont définies comme des vues matérialisés sur le médiateur M.
- La requête Q est réécrite en requête équivalente utilisant les vues (rewriting queries using views)

Architectures d'intégration de données RDF

1. Datasets LOD :
   - chargement de datasets dans un serveur centralisé
   - intégration, indexation et interrogation locale (entrepôts RDF)
   - YAGO, DBPedia, Wikidata
2. Systèmes RDF fédérés :
   - fédération de serveurs RDF (endpoints SPARQL)
   - index centralisé, interrogation distribuée
   - FedX, SPLENDID, Anapsid, Avalanche
3. Protocole « link-traversal » :
   - exploration dynamique du graphe pendant l'interrogation
   - données distribuées, pas d'index, interrogation distribuée
4. Réseaux pair-à-pair
   - réseau P2P structuré (overlay network, DHT)
   - données, indexation et interrogation distribués
   - RDFPeers, Atlas

SPARQL Endpoints

SPARQL endpoint

Service Web (REST) qui exécute des requêtes SPARQL sur un dataset RDF :

- HTTP GET avec requête SPARQL comme paramètre
- résultat en XML, JSON, Turtle, text
- APIs : PHP (ARC, RAP), Java (ARQ-Jena, Sesame), Python (PySPARQL, SPARQL wrapper)

Exemples

- `https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service`
- `http://data.gov/`
- `http://www4.wiwiss.fu-berlin.de/dblp/`
- `http://dbpedia.org/`

Liste : `https://www.w3.org/wiki/SparqlEndpoints`

Problème

Comment décomposer et évaluer une requête SPARQL en utilisant les SPARQL endpoint du LOD cloud ?

Approche « Link-Traversal »

Hypothèses

Chaque ressource (URI) identifie un dataset RDF avec *un endpoint SPARQL*.

Approche

Évaluation d'un motif de graphe par propagation récursive des liaisons de variables :

1. on choisit un motif de triplet avec une ressource identifié (URI) et on évalue ce motif avec l'endpoint SPARQL correspondant ;
2. la réponse retourne un ensemble de triplets avec des liaisons de variables (URI) nouvelles ;
3. chaque nouvelle liaison permet d'évaluer d'autres motifs de triplets par d'autres sources.
4. l'évaluation se termine quand tous les motifs de triplets ont été évaluées

Évaluation similaire à un plan de jointures distribué en boucles imbriquées (sans index).

Approche « Link-Traversal »

Avantage

Évaluation « zero-knowledge » (sans connaissance préalable sur la distribution des données, les sources etc...)

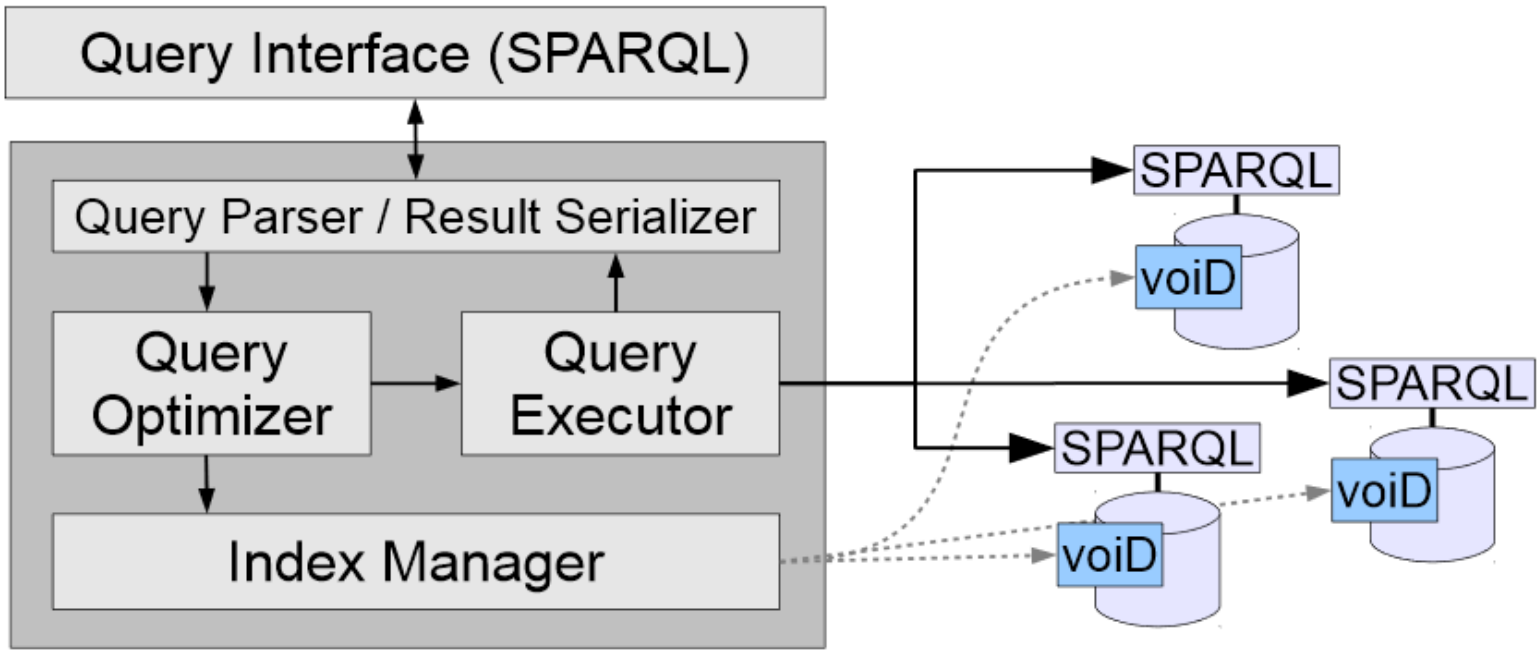
Inconvénients

- **hypothèse forte** : chaque ressource (URI) correspond à un SPARQL endpoint bien identifié
- résultats incomplets et dépendants choix du motif de triplet « seed »
- limité aux motifs qui contiennent au moins une référence vers une ressource (non schéma) ;
- **optimisation limitée** : coût de communication, pas d'ordonnancement de jointures, etc...

3 - LOD : Interrogation

- Architectures d'intégration
- Systèmes RDF fédérés

Système RDF fédéré (exemple SPLENDID)



- catalogue de sources (index manager)
- optimisation de requêtes (modèle de coût) : sélection des sources, décomposition des motifs
- exécution de requêtes (query executor) : jointures distribuées

Catalogues de sources

Catalogue de sources

- Une collection de sources avec la description de leur contenu et de leurs interfaces d'accès aux données (protocoles, formats).
- Les descriptions peuvent être publiées par les serveurs ou collectées dynamiquement par des requêtes SPARQL.

Description de contenu (index)

Pour chaque source :

- index schéma : cardinalités par propriété et classe
- index structure : chemins / sous-graphes fréquents
- index instance : statistiques, histogrammes de valeurs

Les deux dernières types d'index sont coûteuses à générer et à maintenir.

Problèmes

- pas de standards de description de sources RDF
- coût d'acquisition et de maintenance des catalogues

Sélection de sources

Algorithm 1 Source Selection for triple patterns using VOID and `ASK` queries.

Require: I_p; I_τ; $D = \{d_1, \ldots, d_m\}$; $T = \{t_1, \ldots, t_n\}$ // indexes, data sources, and triple patterns

1: **for** each $t_i \in T$ **do**
2: $sources = \emptyset$
3: $s = subj(t_i)$; $p = pred(t_i)$; $o = obj(t_i)$
4: **if** ! $bound(p)$ **then**
5: $sources = D$ // assign all sources for unbound predicate
6: **else**
7: **if** $p =$ rdf:type $\wedge$ $bound(o)$ **then**
8: $sources = I_\tau(o)$
9: **else**
10: $sources = I_p(p)$
11: **end if**
12: **end if**
13: // prune selected sources with `ASK` queries
14: **if** ! $bound(p) \vee bound(s) \vee bound(o)$ **then**
15: **for** each $d_i \in sources$ **do**
16: **if** $ASK(d_i, t_i) \neq true$ **then**
17: $sources = sources \setminus \{d_i\}$
18: **end if**
19: **end for**
20: **end if**
21: **end for**

- I_P : propriétés $\rightarrow$ sources
- I_t : types (classes) $\rightarrow$ sources
- résultat : motif de triplet $\rightarrow$ sources

Évaluation de motifs de graphes I

Algorithme général

- Entrée : un motif de graph $P = \{t_1, ..., t_n\}$ où chaque motif de triplet t_i est lié à un ensemble de sources $D(t_i) \subseteq D$:
- Étapes :
  1. décomposition : générer des partitionnements Π de $P \times D$ en couples (P_i, D_j) où P_i est un sous-graphe connexe (pas de produits cartésiens) et évaluée par la source D_j.
  2. pour chaque couple (p_i, D_j) dans la décomposition Π : évaluer p_i sur D_j et joindre localement avec le résultat intermédiaire
  3. résultat : union des résultats partiels obtenus dans les deux étapes précédentes

Optimisation de requêtes fédérées

Optimisation de requêtes

- trouver des décompositions efficaces (qui calculent le résultat correct)
- chaque décomposition représente un plan de jointure avec des sources distantes

Problèmes liés à la distribution

- coût d'évaluation dominé par le coût de communication
  - réduire le nombre de sous-requêtes évaluées à distance
  - réduire la taille des résultats intermédiaires
- modèle d'estimation
  - précision de méta-données/statistiques limitée
  - jointures : estimation des coûts et des tailles des résultats difficiles (variables dépendantes)

Optimisation : Stratégies de décomposition

Décomposition complète

- évaluer chaque motif t_i sur toutes ses sources $D(t_i)$
- inefficace : beaucoup de requêtes et de résultats redondants

+ résultat complet
+ catalogue simple
- beaucoup de requêtes et de résultats redondants

Décomposition par groupes exclusifs

- regrouper les motifs de triplets qui sont couvertes *par une seule source*
- évaluer chaque groupe exclusive par une sous-requête et les autres triplets

+ résultat complet
- efficacité dépendante de l'existence de groupes exclusifs
- requêtes et de résultats redondants

Optimisation : Stratégies de décomposition

Regroupement par ressources (sujets)

- hypothèse : pour chaque ressource (sujet) il existe une source qui couvre toutes ses propriétés
- regrouper les motifs partageant le même sujet (star queries) et choisir les sources partagées par tous ces motifs

\+ efficace

\- hypothèse forte

Regroupement owl:sameAs

- hypothèse : les sources sont connectées uniquement par des liens owl:sameAs
- regrouper les motifs de triplets (?y owl:sameAs ?z) avec tous les autres motifs contenant la variable ?y et choisir les sources partagées par tous les motifs

\+ efficace

\- hypothèse forte

Optimisation : Jointures distribuées

Objectif

Réduire la taille des données échangées entre le site local et le site distant.

Opérateurs de Jointure distribués

- semi-join : envoyer uniquement les valeurs de jointure au site distant
- bloom-join : envoyer un filtre Bloom qui encode les valeurs de jointure au site distant
- bind-join : nested-loop avec passage de liaisons de variables

Conclusion : Interroger le Linked-Open-Data cloud

1. programmes avec des requêtes paramétrées : boucles requêtes/sous-requêtes
   - \+ données à jour
   - \- inefficace : composition de requêtes ad-hoc par un programme
2. SPARQL endpoint universel qui réunit plusieurs RDF datasets (`http://lod.openlinksw.com/sparql`)
   - \+ efficace : données locales, optimisation SPARQL
   - \- données obsolètes
3. systèmes RDF « fédérés »
   - \+ données à jour
   - \+ optimisation de requêtes distribuées
   - \- mise-en-place complexe
4. systèmes « link traversal »
   - \+ données à jour
   - \+ évaluation « zero knowledge »
   - \+ mise-en-place simple
   - \- résultats incomplets
   - \- optimisation limitée

Bibliographie

- Nicholas Kushmerick. 2000. Wrapper induction : efficiency and expressiveness. Artif. Intell. 118, 1-2 (April 2000), 15-68.
- P. Shvaiko, J. Euzenat, Tutorial on Schema and Ontology Matching, ESWC'05
- Do, Hong-Hai. Schema Matching and Mapping-Based Data Integration, 2006. Thèse de doctorat.
- Stefanidis, Kostas, Vasilis Efthymiou, Melanie Herschel, and Vassilis Christophides. Entity Resolution in the Web of Data, WWW'2014.
- Görlitz, O. (2015). Distributed Query Processing for Federated RDF Data Management.