Setting: Eine Anwendung kennt bereits eine Menge an RDF Daten zusätzlich greift sie auf das Web of Data zu um zusätzliche Daten zu finden und gemeinsam mit den bekannten Daten zu nutzen. Da nicht sichergestellt ist, das die gefundenen Web Daten mit den initial existierenden Daten verlinkt sind, will die Applikation beim Import zusätzliche Links erzeugen.

Components

1.       Die Anwendung, die auf das WOD zugreift

2.       Ein RDF Store auf den die App über SPARQL/SPARUL zugreift

3.       Silk2, das als Server läuft und auf das auch per http zugegriffen wird.

Use Case

1.       Die App stellt Silk2 eine Matching Konfiguration zur Verfügung, die beschreibt, wie  alle (also mehrere) für die Anwendung interessanten Klassen gematcht werden sollen.

2.       Silk2 läd sich alle für die verschiedenen Klassen relevanten Instanzdaten aus dem Store und cacht diese (erst einmal im Speicher, später können wir über andere Lösung nachdenken). ()

3.       Die Anwendung findet Daten im Web, die sie mit den Daten im Store verlinken will.

a.       Sie schickt diese Daten (RDF Beschreibung einer oder mehrerer Ressourcen, eine oder mehrere Klassen) per HTTP an die Silk Instanz

b.      Silk schaut sich die rdf:types der Instanzen an und startet entsprechende Matching Jobs gegen den Cache.

c.       Wenn eine Instanz über dem Ähnlichkeits-Threshold liegt, wird ein Link „Orginal URI owl:sameAs Existierende URI im Store“ erzeugt und für die Rückgabe an die Applikation in einem Cache gesammelt. Gegebenenfalls werden Missing Values im Cache mit Daten von der neuen Instanz ergänzt (zusatzfeature version 2).

d.      Wenn eine Instanz nicht über dem Ähnlichkeits-Threshold  liegt wird davon ausgegangen, dass die Instanz bisher unbekannt ist und daher im Store gespeichert werden soll. Es passiert folgendes: Die Instanz wird in Silk Cache aufgenommen. Es wird ein Triple in den Rückgabe-Cache geschrieben „URI der Instanz silk:matchingResult silk:UnknownInstance“

e.      Wenn alle Instanzen abgearbeitet sind wird das resultierende Triple-Set an die Anwendung zurückgegeben.

4.       Die Anwendung kann jetzt neue Instanzen in den RDF Store schreiben, falls sie das will bzw. die bekannten Daten im Store mit zusätzlichen, bisher fehlenden Property-Values aus den neuen Daten ergänzen, falls Sie das will.

Insert general diagram of silk components (server, instance cache)

Note: A separate instance cache is used for each LinkSpec to improve the matching performance and to allow different blocking specifications.

## Requirements

### Server

1. The Server can be accessed through a REST interface.
2. It provides a interface to commit new RDF Data and to retrieve the matching result.
3. On request, the Server must do a complete matching of all instances in the RDF store and reinitialize the instance cache with all found unique instances.

### Instance Cache

1. Instances are cached in memory. New instances can be added to the cache, but random access is not required.

### Matching Algorithm

1. Accepts one Configuration which may consist of multiple LinkSpecs
2. Incoming instances must be allocated to the correct LinkSpec based on its restriction
3. Instances which do not match a known instance, are written to the instance cache

### Requirements considered for next version

1. If a duplicate instance is found, its data should be merged with the original instance in the cache.
2. Instances are cached on the local/distributed files system

## Implementation notes

### Server

1. Use of JAX-RS to provide a REST interface
2. Accepting new data per HTTP POST in RDF/XML. Two possibilities to return match result:
   1. Directly in the response (if the match takes long timeout may occur)
   2. Response contains a Match Task ID, which can be used to query for the result using HTTP GET
3. Clear the instance cache, create an instance stream and apply the normal update process.

### Instance Cache

1. A new MemoryInstanceCache must be implemented

### Matching Algorithm

1. Already implemented
2. A LinkSpec may contain restrictions consisting of arbitrary SPARQL patterns (not restricted to rdf:type). Jena Model restrictions could be applied to the new instance in order to determine if a given LinkSpec is applicable.

### Next version

1. As the instance values are stored as key value pairs in the cache, the merge should be relatively easy. But random access to the cache is needed to update the instance data, which is also easy as long as the instances are held in memory.
2. Difficult in combination with the previous requirement (random access). Possible solutions:
   1. Using a relational database as backend of the instance cache
   2. Memcached