Cognescent SBI – Semantic Business Intelligence

Sebastian Samaruga
Cognescent
Argentinaa
cognescent@gmail.com

ABSTRACT: *In this paper we addressed the Knowledge Description and Discovery in analytical way. We proposed algorithms where Queries are indexed as knowledge. We supported the possible implementation of the proposed structure.*

Keywords: Knowledge Processing, Query, Open Data, API

Received: 17 March 2015, Revised 16 April 2015, Accepted 27 April 2015

Introduction

Knowledge oriented applications have significance in the Knowledge Description and Discovery process. We built a system in the project we did wherein we indicated the relationship between entities. Our proposed algorithmic statements have queries and well defined structure and we explained it in the paper. Our project repository can able to provide further description.

Scope

Basically, the scope of the project is to provide a product which offers diverse facilities for analysts and developers building knowledge oriented applications. With 'knowledge oriented' we mean any application which could benefit from integration and normalization of loosely coupled data and merging of diverse sources of data Our goal is to achieve this without any previous knowledge of the sources of data, information or knowledge trying to be captured. The system should take care of discovering the identity and the relationships between the entities provided. Also, for such a goal to be met, application designers should not be aware of any specific APIs beyond what are provided in the system.

For this to be accomplished an arrangement of concepts / classes is made and is described below. Algorithmic methods and reasoning are used to perform the inferences needed for being able to describe an application domain or a set of use cases in the way of a unified set of services. Knowledge Description and Discovery (KDD) is used in such a way that analytical, dimensional or rule set evaluation is possible.

The approach taken for trying to make this possible is to use the service (Application described below) as a request – response box. Request may be queries or data. Responses may be query results or posted data augmented with the service knowledge.

Queries are indexed as knowledge. Over this scheme are built the patterns needed for building stateful web applications.

This document only has brief descriptions about the main components of the services.

Further information will be found in the project repositories.

Conceptual Diagram

A conceptual diagram in a pseudo UML notation is depicted below. It contains the main concepts discussed in this document and the relationships between them.

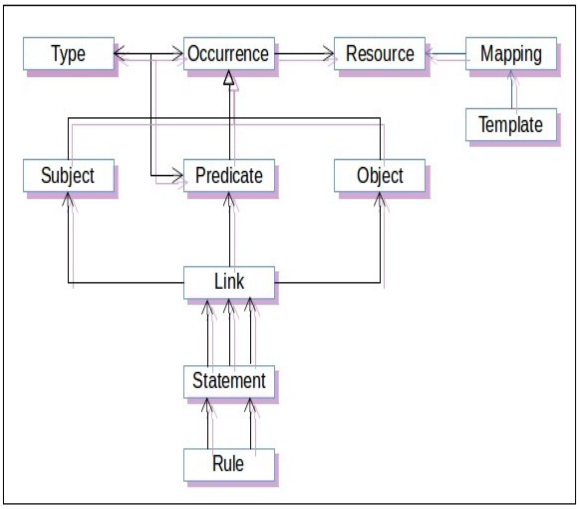

Figure 1. UML Notation

Datasources

RDF / Semantic Web. No need of schema information. LOD (Linked Open Data) via NER (Named Entity Resolution) engine.

Relational (via d2rq).

OLAP (via olap4j connector).

File formats (XLS, CSV, via Apache MetaModel).

XML (Via Apache MetaModel).

Activation: any connector / content type that could possibly be built that outputs RDF: Plain RDF, Relational, OLAP, CSV, Spreadsheets and any possible connectors (JSON for example) as sources of structured data are candidates and some of them have been tested using as an intermediate layer producing XML RDF.

Core Concepts
This are the main concepts or 'classes' in the system.

Resources

A Resource is an instance of an URI (sourceURI) coming from input RDF. Resources are singletons in respect that there is only one Resource for a sourceURI coming from the input RDF.

Any Subject, Predicate, Object of a Link is an instance of an Occurrence which means 'occurrence' of a Resource in that

context. A Resource can have many Occurrences. Beyond the source URI of the RDF statement which gave birth to the Resource (sourceURI) the Resource has the following metadata:

• **SourceURI:** This is the URI of the Resource as is in the source RDF feed to the application, without merging or normalizing.

• **ClassID:** This is a numeric ID assigned to all Resources at creation time coming from a prime number sequence unique to all Resources. Kind of primary key.

• **InstanceID:** This is a numeric identifier which gets 'augmented' with each Resource Occurrence as explained below. Useful for algorithmic inference.

• **NormalizedURI:** Once this Resource is merged/aligned, it gets a new URI which is network retrieveable (by the a service endpoint).

• **MergedClassIDs, mergedURIs:** Upon merging/alignment this is where holds previous Resources history.

• **PredicatesFrom:** Predicates in Links where this Resource is the Subject.

• **PredicatesTo:** Predicates in Links where this Resource is the Object.

Occurrence (SPO)
When a Resource occurs into a Link, its instanceID gets augmented by the following rules. A Resource can play an Occurrence as a:

• **Subject:** Its instanceID is updated by the product of it's instanceID by the instanceID of the Predicate of the Link.

• **Predicate:** Its instanceID is updated by the product of it's instanceID by the instanceID of the Subject and the instanceID of the Object of the Link.

• **Object:** Its instanceID is updated by the product of it's by the instanceID of the Predicate of the Link.

This 'instanceID' makes a Resource 'remembers' where it has occurred and is useful for merging / algebraic inference. The first time a Resource appears its instanceID is its classID.

Links

A link is the representation of a statement triple coming from an RDF source. Link instantiation performs Occurrence (SPO) IDs 'augmentation' and generates metadata for, for example, updating Occurrence Types and the Resource type registry. Also updates knowledge of predicates coming in an out of a Resource (example: being the Resource the Subject or the Object of the Link).

Statements

A set of (three) Links form a Statement in the propositional sense. The statements could be meaningful and they could take part in a (syllogistic) inference chain where any one of the three Links could be inferred from the other two. The section on inference explains better this method.

Rules

Two Statements conclusions may lead to a Rule inference. Using the Links statements and the Statement metadata relationships could be drawn in respect of one being conclusion or cause of another. The inference section on machine learning could be the most appropriate place to take this subject further.

Types

Roughly, Type Inference can be performed regarding which Predicates comes out from a set of Subjects. If a set of Subjects share the same set of Predicates the resulting set could be regarded as an 'inferred' same type. As an example, if I have a set of entities having properties 'price', 'brand', 'discount' I could aggregate the type 'Product'.

Mappings

Mappings are wrappers around Resources designed to extract the most metadata possible from them. There is a correspondence between a Resource and a Mapping but there could be many Resources matching a given Mapping. This is used in the indexing facility mentioned below where it's used for similarity matching.

The first thing a Mapping retrieve from a Resource are its 'roles' (model or schema, predicatesTo). A Resource for a given person can have many roles: 'friend', 'employee'. Any of this roles comprises a set of predicates for which exist a 'metaclass' which is the instance of the Resource in a given role with its predicates (salary for employee).

The Mapping takes care about this arrangement given its 'state' (context). All this is performed using the Resource metadata, its predicates (from and to) and its instanceIDs. For example for considering existing 'employee' role there must exist an statement like: 'employer' 'employs' 'resource'. And, for this to work in every case, inverse relationships must be made and taken into account so every resource has at least one role. Then, predicateFrom(s) Resource properties are used to determine metaclass predicates according the role being evaluated.

Templates

Templates are a set of statements (instances) for a given set of classes which have a given set of properties. They are built mainly from Mappings metadata and for presentation purposes. Imagine a set of results from a query (tabular) or a navigable graph (Template properties are navigable) of entities which must be presented to the user. Content type handling (serialization/ de-serialization) must be taken into account when building applications using Templates.

Transforms

A transformation is, basically, the Mapping of another Mapping. In other words, nested mappings. The result is what both mappings have in common. For example, the source mapping for a 'Theater' in respect to a Mapping for 'Buildings' should contain the roles and metaclasses (see Mappings) 'shared' with both of them
(building.address = theater.address, building.cantFloors = theater.cantFloors)

Architecture

The application design is a REST based request/response set of services which beyond returning a response publish invocation results to 'topics' and 'queues' for further processing.

Registry (Resources)

The Registry Service takes care of Resource mappings and resolution. It also takes care of consume / fetch RDF statements to process, normalize statements subjects, predicates, objects (add new source URI Resources to Registry), creates Links and Predicate metadata.

Index (Mappings)

Apache Lucene backend index. Indexes Mappings metadata. Performs discovery, ID resolution. Performs NER (Named Entity Resolution). Resolves query results. Resolves Mapping keys: inference could be done in which I could possibly assert that two predicates (properties) have the same meaning using learning from the example Mappings at hand and infer they are the same keys
(building.address = edificio.direccion).

Naming (Templates)

URI Resolution services (for normalized Resources). Topics, queues. Publish query results.

Client
Entry point. Main application. Node environment.

Service
The main application REST Services follows:

Where [topicName] is specified, request resolves to that topic. Queues and topics are described below in the section 'Application Runtime'.

Main RDF: /topic/[topicName]|* . * stands for all topics. Indexes request body (RDF). Response dumps (topicName) RDF indexed / augmented. Publish results to [topicName] Useful for SPARQL Endpoints.

REST HATEOAS RDF: /dialog/topic/[topicName]|* . * stands for all topics. Stateful. Dialog. Navigation state. Referrer. Previous posts kept in query context. Send modified response: update / prompt.

DOM: JAXB/JSON Output: /dom/topic/[topicName]|* . toRDF method in response object for CRUD. JAXB Bindings can be built from outputs from the other services.

DOM: OData JS Output: Apache ODataJS. /odata/topic/[topicName]|* . toRDF method in response object for CRUD.

DOM stands for Dynamic Object Model, an internal object representation of the statement graph with classes, objects and properties.

Application level API methods are described below.

Application
Application level API methods. getForm, getState (see below).

Inference
These are a couple of inference mechanisms over Resources and over Mappings.

Formal Concept Analysis (FCA)
This section is regarding ontology merge and data sources alignment over Resources. Given two RDF graphs two FCA relations could be built as having each of them in the object side the subjects of the two graph's statements and in the properties side the objects of the two graph's statements then having relation A and relation B from each graph. The graphs could be built by using graphs properties as axis and Subjects as objects and Objects as properties.

If there exist in relation A an situation such as this object, property pairs apply:
(a, 1); (a, 2); (a, 3) and (b, 2); (b, 3)

And in relation B exists:
(x, 4); (x, 5); (x, 6) and (y, 5); (y, 6)

There is a probability of a being similar to x and b being similar to y.

Semiotic Inference
Inference layer over Resources:
Regarding semiotic triples (sign, concept, object) and mapping them to (Occurrence, Type, Resource) semiotic mappings may be applied to obtain equivalent resources in the way:

sign(concept) → objects

sign(object) → concepts

concept(sign) → objects

concept(object) → sign

object(sign) → concepts

object(concept) → signs

Machine Learning

Proper arrangement of training data with framework data / metadata enables inference of facts for identity resolution and link discovery. Links Statements and Rules are a source for this training data. A proper format for serializing and discover results of learning should be done.

Basic Syllogistic Inference Algorithm

Being Links the representation of syllogistic statements of the kind:

(Socrates, kind, Human)

(Human, life, Mortal)

(Socrates, kind x life, Mortal)

* kind x life = relation product, its ID gets calculated as predicate IDs into a Link triple. And being this labels for the parts of the preceding reasoning:

From = (a, rel1, b);

Rel = (b, rel2, c);

Dest = (a, rel1 x rel2, c);

Then, being '1' the identity relationship, a equals b, we can infer:
From → (a, rel1, b) = (b, rel1 x rel2, c) x (b, 1, b) / (b, rel2, c);

Rel → (b, rel2, c) = (a, rel1 x rel2, c) x (b, 1, b) / (a, rel2, b);

Dest → (a, rel1 x rel2, c) = (a, rel1, b) x (b, rel2, c) / (b, 1, b);

And the Identity resolution formula (a equals c, knowing b):

(a, 1, c) = (a, rel1, b) x (b, rel2, c) / (b, rel1 x rel2, b);

Application Runtime

The application mainly is a REST service listening to the mentioned endpoints. It also can be distributed in a configuration where queues are bound to nodes listening for incoming statements for distribution or federation of knowledge.

API / REST Services

Application level API services are a set of services enabling MVC/DCI object oriented design patterns to be used when developing against the model. The main services where already mentioned. Application design patterns are in development process.

Request Lifecycle

Typically a client would like to index a set of statements, or retrieve another set of statements according a given query. This two cases are basically the same in this model given that indexes are performed in every input in each request.

If the client submitted a set of statements to index them, the resulting output may be an 'augmented' version of the statements

posted. And where a client submits statements to perform a query, it 'augments' the index knowledge trying to use similarity to return some meaningful response.

Queries, Queues, Topics and Names

Publishing to a topic is optional. A default request would return its output (see below) the same way it is made available to the topic. Topics and queues are a facility provided for inter process interaction and application services integration.

Request Output

Queues and topics are basically sets of statements. When publishing to a queue, topic, a Template and its metadata is used as to be able to post the most knowledge available into the given index results (Mappings). Joins between Mappings are made (see Mapping keys) and the Template is augmented.

Application Implementation

The application building upon this model intends to be clean and concise. In a few words, a client requests a form from the application and the application returns the corresponding form in respect to the state the client is in. The client performs its operation in this form and asks the server the next state regarding the form manipulation it has performed. This next state is further manipulated by the client which uses it for asking for the next form.

Use Case Interaction

The development of any application is use case driven following the well known MVC and DCI design patterns. A protocol of view-state exchange is followed for enabling developers build applications in a conversational state manner (HATEOAS REST design pattern). Utility methods are provided and a view/state serialization formats so building frontend frameworks enabled to use knowledge facilities is straightforward.

getForm(state : State) : Form
getState(form: Form) : State

Libraries in any web / scripting language could be build upon this concepts for seamless user driven user interaction. A functional programming style library could be the best bet for working with resources in this way.

It is also possible to use this API and its utility methods (not mentioned here) for building services and service enabled interactions with knowledge discovery and service integration facilities.

References

MVC: https://en.wikipedia.org/wiki/Model–view–controller

DCI: https://en.wikipedia.org/wiki/Data,_context_and_interaction