Semantics & BI

1) Introduction

Features: Repository of semantically annotated data, information and knowledge.

Import RDFized data from heterogeneous data sources. Sync data sources with operations in the model.

Nodes (peers) deployment architecture loaded with the data from each sources, share an Unified URI space (for aligned resources and triples). Functional API for augmentation, discovery and analysis.

Type (classes) / relationship types, relationship instances / state flows (operations, rules, flows, events) inference over 'raw' RDF data from data sources.

Align & Merge. Equivalent resources / triples from different ontologies merged according their meaning. Model by example.

Order inference: inference over the order of events / operations occurred, occurring and that may occur according metadata aggregated from data sources. Enforce use case flows declaratively (model by example)

Uniform API Ports: CRUD bindings to common protocols as REST (HATEOAS / OData), SOAP, LDP (Solid) and others.

Metadata is aggregated decomposing a triple (quad) into their three resources (SPO) and its context. Then a set model is arranged where there is a set for each SPO part, the intersection of the three sets which accounts for the triples itselfs (contexts) and there is another intersection for each of the three SPO combinations.

This last three intersections are regarded as 'Kinds', one for each SPO. So, there is a SubjectKind, an ObjectKind and a PredicateKind. Kinds aggregate 'attributes' and 'values' (see diagrams below) which account for the description of 'classes' and 'metaclasses'. Related attributes describe related classes and related values describe related metaclasses. So, for example, a SubjectKind aggregates Predicates and Objects as attributes and values respectively.

2) Datasources

3) Inference

a) Set predicates

Knowledge aggregated in models should be capable of being abstracted in such a way that general knowledge may be obtained from specific knowledge. Richer query / browsing and inference capabilities should arise from such schema.

```
Data: [someNewsArticle] [subject] [climateChange]
Information: [someMedia] [names] [ecology]
Knowledge: [mention] [mentions] [mentionable]
```

Set oriented approach implementation:

In order to achieve the set oriented abstractions needed (set representation of triples, set models and metamodels) a following (pseudo) API should be implemented:

TripleLoader: Instantiates Resources (SPO, Kinds, Triples) from input RDF.

KindsAggregator: Aggregates and calculates Kind class / metaclass ID URIs. Assignates class / metaclass to SPO resources. Reifies Kinds.

TripleAggregator: Prepares triples from this metamodel layer level as input for an (eventual) next level aggregate.

Model: The model itself (an instance of a metamodel level). Contains set functional arrangements and dimensional arrangements (see below). Base entry point for services API.

Models of data, information, knowledge are relative to their positions respect to other models.

Set: Basic set class. Defined by one set Predicate. Basic set union, intersection, complement operations.

Resource: Superclass of all Set elements. SPO. Monadic wrapper for functional APIs (see below)

Predicate: (Set) Predicate. Holds for a Resource belonging to one Set.

Subject Predicate def: Subject, as Resource appears as Subject in Resources. Is S (& is not P & is not O)?.

SubjectKind Predicate def: attr/val exists at same time in Resources. PO preds holds for subj kind. Is P & is O & Is not S.

Triple Predicate def: SPOs of Triple Resources holds for all SPO preds. Is S & is P & is O.

Hierarchies: when Kinds classes / metaclasses represents some hierarchy relationship the class / metaclass URI ID of them somehow renders this relationship. Then the Kind itself is reified (as an S, P or O) representing this hierarchy's top and aggregating its instances.

Triple Resource Predicate: Context occurs. Context in each metamodel has a meaning and triples sharing the same quad context share meaning being this a temporal, order or causal relationship, among others.

b) Metamodels

i) Kinds aggregation semantics

By the virtue of some resources sharing related 'attributes' and 'values' according their occurrences in multiple triples (for example, given Subjects having related Predicates and Objects) a 'Kind' relationship could be stated of those resources and their types.

Kinds aggregate classes / metaclasses hierarchies given these attributes and values encoded in class / metaclass URI IDs resources. One example could be all the Subjects that 'worksAt' and all resources that 'workAt' 'XYZ Corp.'.

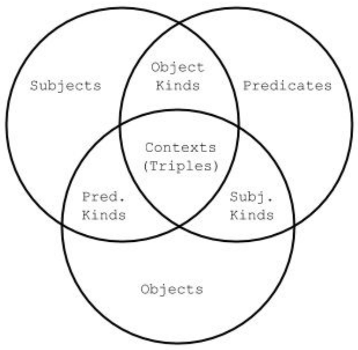
ii) Layers

Models aggregate knowledge in such a way abstractions can be made from source statements and allows for richer query capabilities as in the example triples:

```
Data: :someNewsArticle :subject :climateChange
Info.: :someMedia :names :ecology
Knowl.: :mention :mentions :mentionable
```

iii) Data

SPO Model (Facts)



Occurrence	**Attribute**	**Value**
Subject	Predicate	Object
Predicate	Subject	Object
Object	Predicate	Subject

Triples:

```
Occurrences (Subject ex.):
[context / time] [SubjectURI] [classID] [metaClassID]

Kinds:
[metaClassID] [classID] [attribute] [value]

Contexts:
[context / time] [Subject] [Predicate] [Object]
```

Data model level: this model layer is composed of raw data from which information and knowledge models will be built. Data for this layer comes from the raw RDF from the data sources component.

This set arrangement from triples into SPO and Kinds is the same of the remaining models. Triples feed to the forthcoming levels are aggregated into SPO structure aggregating SPOs, Kinds and triples into new statements. An SPO resource in the next layer triples occur with its corresponding Kind in an occurring triple.

Data layer example:

Triple: Peter worksAt XYZ Corp.

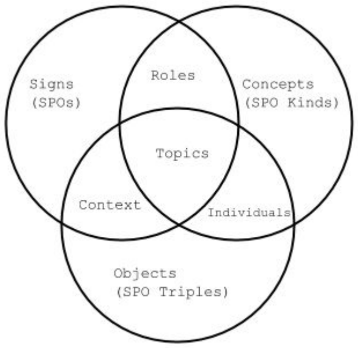
Subject / Subject Kind: Peter / Employee

Predicate / Predicate Kind: worksAt / Employment

Object / Object Kind: XYZ Corp / Employeer

iv) Information

Semiotic Model (SCO, Contexts)



Occurrence	Attribute	Value
Sign	Concept	Object
Concept	Object	Sign
Object	Concept	Sign

Triples:

```
Occurrences (Object ex.):
[context / Topic] [ObjectURI] [classID] [metaClassID]

Kinds:
[metaClassID] [classID] [attribute] [value]

Contexts:
[Topic] [Object] [Concept] [Sign]
```

This model layer will aggregate data into information which can be then converted into a knowledge model. This layer is called ‘Semiotic’ because it is concerned with Signs (SPO resources from the previous model), Concepts (Kinds from the previous model) and Objects (Context Triples from the previous model).

Semiotic layer example:

Object / Role: [Peter worksAt XYZ Corp] (SPO Triple) / Role defs. All Roles that apply.

Concept / Context: Employee, Employeer, Employment / Context defs.

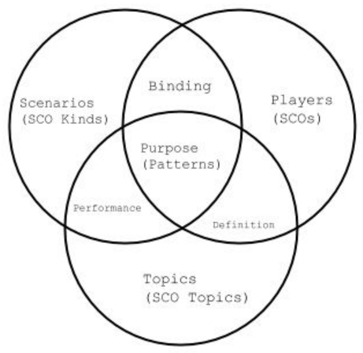
Sign / Individual: Peter, XYZ Corp, worksAt.

Topic: Topic1: hiring. (above SCO triples)

Topics aggregated by same kinds, ordered by SCO contexts (SPO contexts). Order relation examle: hiring, promotion.

v) Knowledge

Behavior Model (TSP)



Occurrence	Attribute	Value
Scenario	Topic	Player
Player	Scenario	Topic
Topic	Scenario	Player

Triples:

```
Occurrences (Topic ex.)
[context / Purpose] [TopicURI] [classID] [metaClassID]

Kinds:
[metaClassID] [classID] [attribute] [value]

Contexts:
[Purpose] [Topic] [Scenario] [Player]
```

This model layer will render knowledge extracted from aggregated triples of the previous data and information layers.

Knowledge layer metamodel example:

Topic / Binding: Topic1

Scenario / Definition: newProject (SCO Context)

Player / Performance: aProject (SCO Concept)

vi) Metamodel example

Metamodel features example:

A of B is C. (S of Triple is Kind)

Peter worksAt XYZ Corp. : EmpTriple

SPO Metamodel:
SubjectKind: Employee(worksAt, XYZ Corp.)
PredicateKind: Employment(Peter, XYZ Corp.)
ObjectKind: Employeer(Peter, WorksAt)

SCO Metamodel:

SignKind: Instance(Employent, EmpTriple)
ConceptKind: Context(Peter, EmpTriple)
ObjectKind: Role(Peter, Employee)

The ‘functional’ notation used above is not casual. It will be used later in APIs designed to build Template(s) or ‘patterns’ which will be the basis for provide a Services interface for interacting with the models.

c) Sets, Monads, Higher order functions

Resources: Higher order functions mappings. Resource monads.

Kinds. Kind monads.

A of B is C. (S of Triple is Kind)

Resources, Kinds: Container, Container Profiles (LDP / Solid).

Bound functions.

Templates are the basic IO messaging method of the service interface which allows for CRUD and state flow (rules, flows, events) manipulation.

Functional Template:
Template : (Kind, Template lhs, Template rhs)

Template example:
Employment(Person((Age, 'young') {&/|} (Sex, 'M')), Business((Size, 'Small')))

Template Promotion:
Person -> Employee (at Employment Kind, due to predicate/object statements addition. Add inst. rel/attrs: salary, position, dept, etc. for class via callbacks / prompts of their values or value Kinds, ie.: high salary).

Functional assertions are bound to order relationships encoded in metamodel triples / quads contexts so, for example, one could query about available templates and possible values regarding this state.

Monads:

interface M<T> : public(T) : M<T>

function unit<T>(val: T) : M<T>

function val<T>(m: M<T>) : T

function bind<T, U>(inst: M<T>, transform: (value: T) => M<U>) : M<U>

ResourceMonad<R extends Resource>(resource : R)
ResourceMonad.Triple
ResourceMonad.Context
ResourceMonad.Subject
ResourceMonad.Predicate
ResourceMonad.Object
ResourceMonad.SubjectKind
ResourceMonad.PredicateKind
ResourceMonad.ObjectKind

Bound functions:
Inference, Triple joins: (S -> Object: Kinds, S -> Concept: Triples, Kind -> Sign: Triples)

Monadic type ctor.:
ResourceMonad<T extends Resource>

Subject example:
SubjectMonad extends ResourceMonad<Subject>

Unit function:
if(subjectPredicate.holds(val)
return new SubjectMonad(val);

Bind function:
Specific to each type. Retrieve argument callback.
Returns type monad

d) Dimensional arrangement of Resources

For ease of development in what to ontology alignment and merge is concerned a ‘Dimensional arrangement’ of the underlying data is attached to each model thus providing enough metadata for equivalence resolution and for augmenting functional APIs behavior.

Object example: below is an example of such an arrangement for a given Object. It consists of a ‘dimension’ of the object types being considered, a ‘unit’ of measure in such dimension and the instances of their ‘values’

Dimension: Object Kind.
class: hiers (attributes) / ranges: meta (values)

Unit: Predicate Kind.
class: hiers (attributes) / instances: meta (values)

Value: Subject Kind
class: hiers (attrs) / domain: meta (values)

Model representation:
Map<Dimension, Map<Unit, Set<Value>>

Dimension, unit, value: hierarchies, primitives. Primitives and composition should be used for ontology translation / merge.

SCO Primitives, mappings, merge.
Knowledge level similarity. Primitive Signs, Concepts, Objects.

Function<Domain, Range> : instances.
Monadic bound functions. Augment functional API.

i) Equivalence inference functions

Model representation:
Aggregate Model Kinds (Subject example):

Map<Subject<Map<Predicate<Map<Object, ClassMeta>>>

ClassMeta: Kind Resource. URI ID. Reifiable. Resource resolution. Predicates (attribute / value)

Align & Merge: mappings between equivalent Subjects, Predicates, Objects (merge dimensions, units & values):

Eqivalence sets:

Set<Set<Subject>> : ranges.
Set<Set<Predicate>> : instances

Set<Set<Object>> : domains

Equivalence functions SPOs:

Identify keys (attributes/values) that doesn't repeats for the same occurrence class.

Equivalent Predicates: equiv. domains / ranges, instances mappings.

Predicate equivalence predicates having equivalent SO in their statements.

e) Type inference (Subjects, Predicates and Objects)

Type inference is performed via Model Kinds and dimensional metadata.

f) Equivalence inference (Resources and Triples)

Equivalence of resources and triples is meant to be the basis for ontology alignment and merge of, for example, diverse vocabularies talking about the same subjects.

Type inference and dimensional arrangements are to provide the necessary means for performing such a task.

g) Relationship type and instances inference. Graph navigation

Due to a Subject having a class / metaclass of a given Kind there could be reasoning about the rest of the relationships and the values the Kind and the Subject relations could have.

h) State transitions (Rules, Operations, Flows) inference

Contextual order relations lead to a tree structure in which when given some facts (statements) occur in determinate position in a given sequence in the tree they fires rule, flow and event listener events.

4) Architecture

a) Loaders (Data sources, sync)

b) Runtime (Peer)

i) Input Jena Model

ii) Models (Sets, Dimensions, Tree)

iii) Index

Index: Any Triple to Name(s) Graph fragment. Graph API (Models). Registry bindings (topic/queue) dataflow.

iv) Naming

Naming: Parse / normalize names / URIs (Name entity: domain, NS, parts. Uniform Names abstraction layer. NLP. Dictionary. Definitions. Synsets (equivalence).

v) Registry

Registry: Hierarchical endpoint, dataflow placeholder (possible individuals) in dialogs over purpose protocol. Naming references resolution. Feed.

Listeners: dataflow.

vi) Output Jena Model

vii) Output DOM Model

DOM Model (for ORM like bindings)

Model
types : Type[]
entities : Entity[]

Type : Entity
name : string (URI)
properties : Type[] (Map<string, Type>)
entities : Entity[]

Entity
name : string (URI)
type : Type
properties : Entity[] (Map<Type, Entity>)
payload : object

c) Services API

i) Templates

Template definition:
Template : (Kind, Template lhs, Template rhs)

getTemplate (Template state) : Template next

putTemplate (Template temp) : Template next

Contextual ordering.

ii) Listeners

Flow listeners, Rule listeners, Event listeners. Events regarding tree statements context traversal.

d) Functional query API

‘Nodes’ augmentation, analysis and discovery. DCI (Data, Context, Interactions) design pattern: behavior model. Model by example.

1. Agent client interfaces ‘activated’ w./ domain behavior. Components. Template state.

e) Ports (Representations)

i) RDF(S) / OWL

RDF(S):
Kinds: Classes. Props: classId, metaClassId, attrs, values.
SPOs: Instances of Kinds. Props: ctx, resourceUri, classId, metaClassId.
Triple class. Triples: reified statement.

OWL:
Kinds: Classes. Restrictions.
RDFS Props.
SPOs: Individuals.
Triples: Individuals.

Add Inference layer.

ii) SPARQL

iii) RESTFul HATEOAS (OData)

iv) SOAP

v) Solid / LDP

Solid:

WebIDs / WebID Profiles: Concise Bounded Description.

Container WebIDs: Data (SPOs)
Container Profile WebID: Schema (Kinds)

(Persons, Organizations, Groups, Devices, Requesting 'Agents', Server, Service: WebIDs Profiles?)

Definition
Given a particular node (the starting node) in a particular RDF graph (the source graph), a subgraph of that particular graph, taken to comprise a concise bounded description of the resource denoted by the starting node, can be identified as follows:
. Include in the subgraph all statements in the source graph where the subject of the statement

is the starting node;
. Recursively, for all statements identified in the subgraph thus far having a blank node object, include in the subgraph all statements in the source graph where the subject of the statement is the blank node in question and which are not already included in the subgraph.
. Recursively, for all statements included in the subgraph thus far, for all reifications of each statement in the source graph, include the concise bounded description beginning from the rdf:Statement node of each reification.
This results in a subgraph where the object nodes are either URI references, literals, or blank nodes not serving as the subject of any statement in the graph.

Representations: Content negotiation / Activation.

Identity / Discovery (WebIDs, Profiles)

Authentication (WebID-TLS) / Login (HTML5 keygen cert. pub.)

Contacts Management

Messaging / Notifications

Feed aggregation / Subscription

Comments / Discussions

Friends / Followers / Following lists (topics, profiles). Users / Agents (event flows). WebIDs.

LDP: Linked Data Platform: RESTFul applications, shared storage space.
Resources, CRUD (Containers), Drive (Gestures, Augmentation)

Servers: LDP Implementations (LDNode). Decentralized RDF Data Model.

Basic Container -> Direct Containers (Multiple Facets)

Solid SPARQL: Each Resource is its own endpoint (default graph) INSERT, SELECT, DELETE

JSON-LD: (Metamodel loaders, services: OData). Contexts: Declarative schema.

WebSockets: Pub / Sub. Listeners changes.

Notifications (LDN Linked Data Notifications w3c.org): Inbox, discovery.
ActivityStreams (w3c.org). Messages / Streams.

POD: Personal Online Datastores:

. Server: Impl Default Containers (Kinds, Purposes). Workspaces, Preferences.
. Clients: Activation, Dashboards (workspaces, AngularJS DOM Activation).
. Applications: Configuration (Purpose instances). Runat Server / Clients.

Metamodel / Augmentations (Enhancements: Sling, Stanbol)
Kinds: Containers (Direct Container, mult. Facets).
. Container WebID Profile: Kinds Schema.
. Container WebID: Schema instances.

. Container schema / instance browseable, linked (HATEOAS).

Resource: Address, Type, Representation. Dereferencing.

Naming, Index storage, Registry: Persistence, Models.

f) Agents

   i) Activation (over Representations)

   ii) Client API configurations.

5) Lab

a) Encoding and addressing

b) Octal order relation encoding

c) Lab: Higher order like predicates for SPOs, Kinds, Triples aggregates. Monadic constructors / wrappers. Logic, sets, filters, selection. Algebra (Monadic functors)

d) Lab: NodeJS + node-java or messaging protocol (JSON + Jersey / JMS). Browserify, local peer’s nodes.