Copyright (C) 2017 Sebastian Samaruga.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU Free Documentation License".

Business domain translation of problem spaces (WIP draft)

© 2017. Sebastián Samaruga (ssamarug@gmail.com)

Abstract. Keywords. Introduction (business domains integration use cases). Schema merge and interoperability.

Streamline and augment analysis and knowledge discovery into enhanced declarative and reactive event driven process flows. Bridge the gap between frameworks, protocols and tools for building ‘real’ Semantic Web backed (Big) data applications.

Abstract: The concept revolves around a network of (perhaps existing network’s) Peers which interacts in the aims of fulfilling business Objectives (Task flows) for which Peers are selected to perform regarding their Profiles and a given Purpose they and Objectives are proposed to accomplish. A Purpose is a kind of ‘abstract goal’ to be met. Profiles are the (maybe evolving) capabilities for the resolution of Objectives problem kinds.

Given business domains ‘problem spaces’ a ‘translation’ should be made which encompasses given a set of problem Objectives to be solved in one domain (maybe because of Events in that domain) another set of co-requirements in other domains which triggers new Objectives into the flow which shall be accomplished for the global Purpose to be met.

An example: In the healthcare domain an Event: flu diagnose growth above normal limits is to be translated in the financial domain (maybe of a government or institution) as an increase of money amount dedicated to flu prevention or treatment. In the media or advertisement domain the objectives of informing population about prevention may be raised. And in the technology sector the tasks of analyzing and summarizing statistical data for better campaigns must be done.

The technology empowering such endeavor must not be other than that of the graph-oriented featured semantic web paradigm, with tweaks into functional programming concepts and Big Data / Machine Learning inference providers.

Besides some initial type inference over RDF datasets the features here discussed not only focus on ontology engineering or modeling. The aim is more like to provide an engine (some like DOM / JavaScript is for HTML) for whatever RDF can be provided for it from whatever sources, not only DBpedia but seamless integrating RDBMSs / REST / LDP or any other backends / datasources / protocols.

Considerations

Although the proposed application mentioned in this document is thought to be implemented in an end to end full stack manner (from presentation through business logic to persistence) the core components meant to be used here are distributed and functional in nature.

Implementations of the framework shall allow for the discovery and publishing of profile driven endpoints.

As far as I know, RDF / RDF(S) or OWL are not currently widely adopted in the enterprise, at least in the pervasive manner I think they should, in favor of more traditional methods of information storage and retrieval such as RDBMSs and, also, NoSQL datastores. This seems like the benefits of 'semantic' datastores are only visible to a selected group, seen like kind of 'gurus' by ‘relational’ guys, and that no mortal will be able, or even needing, to take advantage of SW at all. This is sad because I consider SW as *the* ideal platform for Big / Linked Data business and integration endeavors.

There must be a better way than an 'all or nothing' approach where one paradigm takes over another. Simple CRUD / ERP, BI and even Enterprise Application Integration (ESB) problems currently lack of 'ontologies' in favor of 'schemas' or interfaces when the former could be of great help when managed from the governance perspective of an organization.

Having a 'kind' of semantic repository, which is aware of all 'triples' the integrated systems 'produce' (Adapters), has previous knowledge of the 'world' (augmented/loaded with domain ontologies), performs type inference, ontology merge and alignment, resolves inferred links from new knowledge, performs ordering of knowledge regarding contexts sorting criteria and aggregates this knowledge into layers: from raw data to symbolic statements to (inferred) behavior statements, may be the foundation (via a functional API) of a Master Data Management (MDM) like 'semantic' component.

Proposed implementation details of the component discussed below involve many low level concepts which are specific to this attempt of data integration and are not necessary 'orthodox' SW practices. In the end, the component offers a set of adapters/datasources, ports/endpoints for different protocols/services with similar semantics so it can be used by many clients as possible as a 'MDM Hub'.

Features: Repository of semantically annotated / aggregated data, information and knowledge integrated from diverse sources and consolidated into reactive declarative meta models for enterprise class application services backend containers.

Features

An important goal will be to be able to work with any given datasources / schemas / ontologies without the need of having a previous knowledge of them or their structures to be able to work with them via some of the following features:

Achieve plain RDF serialization to-from any service / backend through Node Bindings (below). Ontology (schema) merge. Temporal, order (in contexts), Identity, Type and attributes / relationships alignment / augmentation.

Model statements into layers, each one having its own 'abstractions' for the roles each statement parts play:

Data layer: Infer type information, Classes / Kinds roles. Properties (domain / range role Kinds). From facts layer statements (data) infer which property kinds holds for which domain / range role kinds:

Who is a Person.
Who is a Business.
Which relations (properties) holds between them.

Information (schema) layer: Events which entail Flows given certain statements (relations) roles.

aPerson (hired), worksFor, someCompany (hires);

Knowledge (behavior) layer: Given certain Rules some Messages are ‘activated’.

betterJobOffering, employeeBehavior, jobSwitching;

Knowledge, information and data behavior (CRUD) entails ‘propagation’ into upper / lower levels of statements updating them accordingly via dataflow graphs (below).

Data example: aProduct, price, 10;

Information example: aProductPrice, percentVariation, +10;

Knowledge example: aProductPriceVariation, tendencyLastMonth, rise;

Features should be provided by the framework / datastore (alignment):

Align identity: merge equivalent entities (with different URIs). Align types (schema / promotion: infer type due to role in relation).

Align attributes / links: augment knowledge (properties and values) about entities (because of type / role alignment).

Align ordering: sort entities regarding some context / axis (temporal, causal, composition and other relations).

Implement functional query / transformation language: for a given entity / concept (Monad) being able to browse / apply a function which entails some other result entity (Monads).

Resource endpoints. RESTFul interfaces for metamodels interaction. Provides / consumes (feeds / streams) events / messages declared via dataflow engine. The datastore should be a HATEOAS web application.

Implement XSL Driven declarative dataflow engine: Streams 'pipes' declaratively stated in XSL for reactive behavior definitions. Functional query / transform semantics. Datastore application publish / subscribe to this (request / response 'filters').

Implement a 'Node' abstraction: Integration of diverse (wrapped into metamodels) datasources / backends / services / protocols via the implementation of service contracts and exposing a plain RDF IO (dialog protocol) interface. A Node Binding to other Nodes shall allow to integrate via dataflow semantics diverse datasources / datastores.

Provide discovery services for data / schema / behavior (bus / stream events actors and roles). Bind Profiles by ontology metamodel alignment: Registry: Ordering alignment, Naming: ID and instance matching alignment, Index: Links and attribute alignment. Big Data and EAI integration patterns. Reactive adaptive dataflow containers (criteria instead of hardcoded addresses).

Topics:

Nodes

Server / Client Peers. RDF statements IO / ETL syndication / sync to - from any backend / datasource / service / protocol. Bindings: Engine Peers wrap exposed protocols. Schema aware 'less'. Seamless merge diverse datasources and schemas keeping them in sync.

Services: Nodes with IO (registry, index, naming, ML, Big Data, reasoner, BRMS, etc) provider Nodes.

Discovery by profile: criteria / patterns instead of endpoint addresses. Referrer contexts. DCI. Bus: dataflow streams (pipes) actor / role pattern.

Metamodel

Reification: types, layers.

Entities (Resource / Statement): Type / Data (Class / Kind), Information (Event / Flow), Behavior (Rule / Message).

Type inference

Kinds (occurrences). Classes (aggregation).

Layering

Aggregation. Data, information, knowledge.

Alignment: ID Merge

Type / equivalence augmentation. Kind / Class aggregation. (Data layer).

Alignment: Order, contexts

Contextual comparison augmentation. Rule / Message aggregation. Roles (Behavior layer).

Alignment: Attributes / Links

Property / value augmentation. Event / Flow aggregation. (Information layer).

Functional (query / transforms) API: Monadic. Reactive dataflow (distributed nodes activation graph). IO format: Resource metamodel hierarchy.

Terms: internal upper ontology aligned. Variables, placeholders. Pointers: resolve subject in role in context (Semantic pattern IDs).

Node reactive dataflow integration

XSL Declarative reactive streams: DCI (Message, Subscriptions, Processor). Functional transforms declarations plus Terms. Events pipes / routing. Resource ‘activation’: layers / reified Nodes propagation. Actor / role. Reified Resources into transforms (dynamic / patterns) addressable. DCI dynamic behavior. Pattern based IO. Cactoos.org.

Nodes dataflow integration (message / event driven augmentation). Service Nodes.

RESTFul Metamodel layers Resource hierarchy endpoints: XSL filters / activate inputs and outputs. Generates derived (activated / augmented) statements. Activates functional layers aggregation.

HATEOAS: JSONLD / HAL.

Engine Node: Generic client (Peer, browser). Transforms (wraps) Node into protocols / schemas. ‘Semantic ORM’ (Dynamic Object Model, Actor / Role DCI, JAF). HTML DOM / JS like platform bindings. Declarative discovery / activation. Reactive platform bindings (Java, JS, etc) over RESTFul Resource endpoints.

Business Domain Translation of Problem Spaces: TBD.

Generic client (browser): Peer configuration, discovery, ETL. Dashboard (default app: CRUD / Social / Process workflows, etc.).

Application Demo

Dashboard (CRUD enabled) of aligned / merged syndicated data sources. Dynamic analysis / mining / drill generation. Inferred business use cases (domain driven development) frontend.

Dashboard. Admin. Features (integration). Domain translation. Alignment (merge) Actionable items (dataflow, activation, drill, mine, browse). Stubs other features.

TBD.

Architecture

Node (Metamodel + XSL / Functional + Message API): Apache ServiceMix (like) archetypes (RX Camel).

Node interface(s). API impl. Metamodel / XSL / Functional / Messages:

Node Binding archetype: Implement Node for exposing data sources / Node syndication / bindings (RDBMSs, services, protocols, formats, endpoints, etc.). Synchronization.

Node Augment archetype: Implement Node interface for knowledge enrichment (data enhancement / linking / augmentation, reasoning, inference, learning, etc.). Default Augment Nodes: ID Alignment, Order Alignment, Attribute / Link Alignment, Index, Registry and Naming Services.

Node Client archetype: Implement Node APIs for exposing client infrastructures. REST / SOAP / SPARQL Endpoints (default via AOM / DOM). Web Application client (framework). Others.

Message flow (incomplete):

Transport: Nodes setup / discovery / dataflow bound by declaratively composed pipes in XSL (patterns over bus / queue / topic endpoints). API ‘callbacks’ (reactive typed / bus pattern based publishers / subscribers). Dialog / Flow Messages (IO: variables, wildcards, placeholders). Example: Bindings posts its ‘schema’ on creation (Flow Message). Augmentation / Client eventually ‘ask’ Binding for ‘data’ in its schemas. Client gets ‘augmented’ Flow(s) for its requests. Nodes keep their Metamodel(s) updated (with references to other Nodes models / resources). Hypermedia (REST HATEOAS: HAL / JSONLD) Messages holds links / patterns (discovery) to other Nodes model resources / Messages. TBD.

TBD.

AOM Functional DOM. Data / Context (schema, roles) / Interactions. Dynamic Object Model. Data. Vars. Placeholders. Model (infer from input / output).

Message: Reified metamodel statement: Resource, Kind, Class, Event, Rule, Flow. Message encoding (the most elemental Flow is a Subject having an Object as its Property).

Potential encoding of nested reified Metamodel message (Flow):

(Flow (Rule (Event (Class (Kind (Resource))))));

XML like with nesting structure / repetitions / links / references (for XSL declarative pub / sub pipes).

Encode (multiple) quads attributes / values (for a resource with the same parent).

Update Metamodel for nesting / encoding via quad players occurrence.(TBD. Example):

(Event, Class, Resource, Statement);

(Class, Kind, Resource, Resource);

(Kind, Resource, Statement, Class);

Flow encoding: order, contexts encoding.

Rule encoding: schema (attributes / links) type promotion encoding.

Event encoding: Data identity encoding.

Nodes: retain 'dialog' state (Metamodel + Messages) vars / placeholders. Message 'model': data, schema / transforms, behavior. Lambda (server less) ‘runat’ features. Activation graph.

Messages: encoding. 'query' part, 'assertions' part (relative to other Nodes in respect to their Metamodels. Message semantics. Message 'model': data, schema / transforms, behavior. RDF materialized reified metamodel.

Node: Aggregated Metamodel + XSL (declarative pipes) over discoverable (templates) reactive Resource endpoints (in / out Nodes). Interface methods (as templates in XSL over metamodel + Functional API) + Message logic for processing request / response. Declarative 'request / response': criteria over publish / subscribe. Filter messages.

Node: Bindings Node. Discover if Node has knowledge regarding Message. Emit knowledge regarding Message (syncing if necessary). Apply Metamodel + XSL transforms. Endpoints 'wrapping' datasource / protocols CRUD (Message encoded semantics determine operations, criteria and arguments).

Node: Type / ID Align / Augment Node.

Node: Order / Contexts Align / Augment Node.

Node: Attribute / Link / Promotion Align / Augment Node.

Node: Index Services Node.

Node: Naming Services Node.

Node: Registry Services Node.

Node: Client Node.

Backend Nodes (features): Layers (types, aggregation), Alignment (id merge, order, links / rels), Functional query / transform (as Resources), XSL declarative pipes, REST Resource endpoints (pipes / events / message aware: methods). Each Node declaratively listens / augments other Nodes.

Nodes: Bus (syndication) Message driven activation / dataflow (pluggable discoverable features: declarative 'Nodes' activate on inputs, augments meta models, bus IO). Bindings: nodes implementing backends / protocols / datasources IO (sync).

XSL Templates: declaratively composed monadic (selectors / query / pattern) functional transforms.

Browser: Client Node (over protocol Node). Engine Node API: features MVC / DCI DOM / Functional REST metamodel. Engine Node declaratively activates on metamodels (Node IO). Renderers (UI / protocol Bindings) implements Node. Engine: local stub.

Dashboard. Admin. Features (integration). Domain translation. Alignment (merge) Actionable items (dataflow, activation, drill, mine, browse). Stubs other features.

Resource URI scheme / format for proper feature / activation patterns. Naming, index, registry features (used by alignment features pipes). Express features as XSL declarative templates? (Feature / Node object model / interfaces).

Node Protocol

Dialog. Reactive dataflow (variables / placeholders).

TBD.

Service Nodes

Index

Naming

Registry

Alignment / Augmentation

Inference / Reasoners

Bindings. Node syndication

Potential datasources / sync (syndicated / merged) Nodes includes (but are not limited to):

- RDBMSs sync. JDBC RDF IO.
- EAI Bindings (CRM / ERP / ESB RDF IO Bindings).
- Web Services (SOAP / REST IO sync).
- DOM / DCI (Semantic ORM, Dynamic Object Model / Data Context Interaction patterns)
- OData (as of Apache Olingo)
- JDBC Driver / Node (in-mem dynamic generated schema)
- Java EE Bindings: JAX-RS, JAX-WS, JMS.
- OWL upper ontologies (ISO15926).
- OWL / RDF(S) SPARQL Node. Reasoners. Sync.
- Index (services).
- OMG MOF (MDA).
- ESB Bindings (Apache ServiceMix / Red Hat Fuse / Mule ESB).

- BI: Business Intelligence dimensional analysis / mining provider. Master Data Management. Governance.
- Business Rules / BAM / BPM / Workflow engine bindings (JBoss KIE).
- Atom / RSS syndication.
- RESTFul endpoint generation. Reactive functional asynchronous. HATEOAS (HAL / JSONLD).
- LDP / SoLiD.
- Content repositories (JCR / WebDAV).
- ML: TensorFlow.
- JXTA Nodes. P2P Discovery.

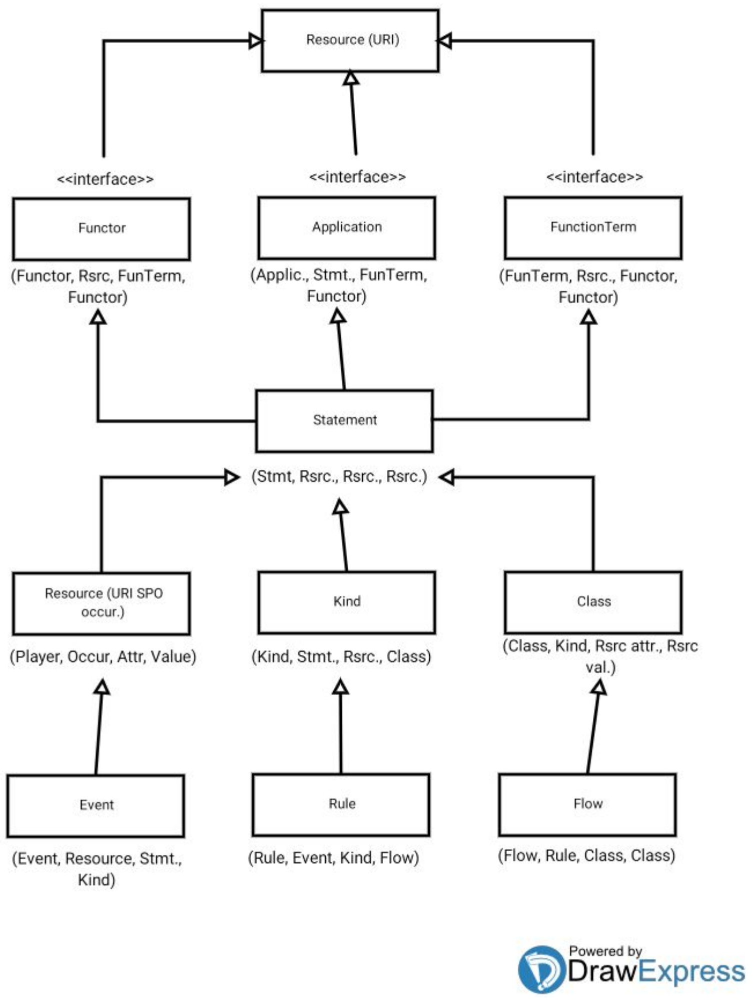
Client Bindings (Engine)

Bindings in Nodes playing a ‘client’ role may be consumed by other Nodes as ‘provider’ ones. Those Nodes may provide ‘transform’ augmentations (queries, reports) as reified entities / schema available to other Nodes for consumption.

- Semantic Web Browser: from documents (DOM) to applications (AOM). Metadata (declarative descriptions, reactive dataflow). Frontend / protocol activation (AOM metamodel). UI ‘renderer’ / Protocol ‘implementor’ implements Node interface. User gestures implemented as protocol ‘dialog’ (vars, placeholders). AOM: Semantic ‘ORM’ DOM + DCI + Functional API.
- Platform Bindings: JAF like aware language platforms implementations. Deployable (clients / services containers).

Metamodels

Models reification. Models encode models. CSPO Resource reification.


Resource (URI)
<<interface>>
Functor
(Functor, Rsrc, FunTerm, Functor)
<<interface>>
Application
(Applic., Stmt., FunTerm, Functor)
<<interface>>
FunctionTerm
(FunTerm, Rsrc., Functor, Functor)
Statement
(Stmt, Rsrc., Rsrc., Rsrc.)
Resource (URI SPO occur.)
(Player, Occur, Attr, Value)
Kind
(Kind, Stmt., Rsrc., Class)
Class
(Class, Kind, Rsrc attr., Rsrc val.)
Event
(Event, Resource, Stmt., Kind)
Rule
(Rule, Event, Kind, Flow)
Flow
(Flow, Rule, Class, Class)
Powered by DrawExpress

The main idea is to be able to merge diverse data sources (from existing applications databases for example) and from they and their metadata expose 'declarative' application models which can be used for domain driven front ends or services.

Metamodel (AOM: data Application Object Model) should be to RDF / Semantic Web as DOM (Document Object Model) plus JavaScript is for HTML / XML.

A model loads/aggregates resources from statements or from other models.
Statements are the means for obtaining resources (SPOs). In the context of an statement a resource has a ‘kind’ (type or class). For example, a subject (S) has a ‘subject kind’ (PO of an statement). Given any resource occurring in a statement its kind will be the remaining SPO parts of the statement, being one part the ‘attribute’ (class) and the other one the ‘value’ (metaclass).

Layers, Resource flow, aggregation: ‘layers’ are built by reifying Statements from one level as Subjects of another level, Kinds as Predicates and SPOs as Objects (data, information, knowledge). TBD.

Resources: Statements, CSPOs, Kinds: encode ‘roles’ (metaclasses) of a Resource occurring into an Statement.

Transitions (dataflow transforms) encoded as Events / Rules / Flows. Type promotion, attribute / links augmentation, order alignment, equivalence augmentation.

Events (goodEmployee) are aggregated from Resources (and are a subclass of ‘reified’ resources).

Rules (goodEmpPromotion) are aggregated from Kinds (and are a subclass of ‘reified’ kinds).

Flows (promotion) are aggregated from Classes (and are a subclass of ‘reified’ classes).

Input Statements matches Event patterns from which Rules activate firing Flows transforms.

Functional API via Statements: TBD.

Quads encoding: contexts, reification, layers. Hierarchies (order, class / instance) layers. Naming, links / attributes. TBD.

Node IO: a Node internal metamodel may perform alignment, aggregation, augmentation, inference or other features over its (RDF) inputs as a consumer. As a producer it should expose its (acquired) knowledge only as plain RDF again observing a simple protocol for CRUD / query / browsing (REST HATEOAS, HAL / JSONLD)? so any other Nodes may ‘plug’ (listen / publish to) any other Nodes streams. This without being aware of other Nodes internals (pipes, transforms, aggregation, syndication, etc.)

Type inference

Kinds (occurrences). Classes (aggregation).

A special type of Resource are 'Kind’s. Each SPO in a Statement has its corresponding Kind which has an ‘attribute’ and a 'value’ in respect to its position in the triple. For example, the triple:

(Peter) (worksAt) (IBM)

Has a 'SubjectKind’ of (worksAt, IBM) for its Subject (Peter). Subject’s attribute and value are (worksAt) and (IBM) respectively. This metadata could be used for basic type inference by aggregating ‘Employees’ (worksAt domain) and specific employees (which work at IBM) or ‘range’ (metaclass).

The same holds for classifying Predicates and Objects into their types and their corresponding meta-types. Different Model layers (discussed later) have its own (Predicate based) SPO Sets definitions and, thus, their own Statement and Kind structures.

Kinds reification may be performed given, for example, this example 'Employee' SubjectKind becoming a 'Employee' Subject. This way attributes and links may be stated for the 'Employees' set in general (Grammars model).

Resources (SPOs) within an occurrence into an Statement have a Kind (type) corresponding to the Resource's attribute and value (other SPOs of the Resource).

For example, a Subject 'John Doe' has a Subject Kind of 'Employee' in the Statement:

'John Doe', 'worksAt', 'someCompany'.

Kinds aggregate Classes and may represent compound types (many attributes sharing their values.

Predicates and Objects are classified the same way than Subjects.

Resources may have multiple occurrences, as subjects, predicates and objects. Regarding Kinds, for example for a given Subject, it SubjectKinds will be the set of all Predicate attributes and Object values according their occurrences in triples where there is that Subject (the set with kinds attrs/values intersection is populated from source triples correspondingly). Then aggregation is done for class / metaclass inference.

Kind hierarchies are given being a subclass having a superset of the attributes of its superclass and given an identification mechanism for Kinds which allows to infer the relationship.

Layering

Data, information, knowledge.

Data, information, knowledge: price, price variation, price tendency.

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]
```

Model Layers:

The first Model (Facts) is built from raw input triples (SPOs) and its Kinds are aggregated according its Statements and Resource occurrences.

The first level: data (facts) deals with ‘pure’ source RDF CSPO statements. It corresponds with the management of ‘signs’ (SPOs URIs) and the semiotics branch of ‘syntax’ (the study of signs and their relationships).

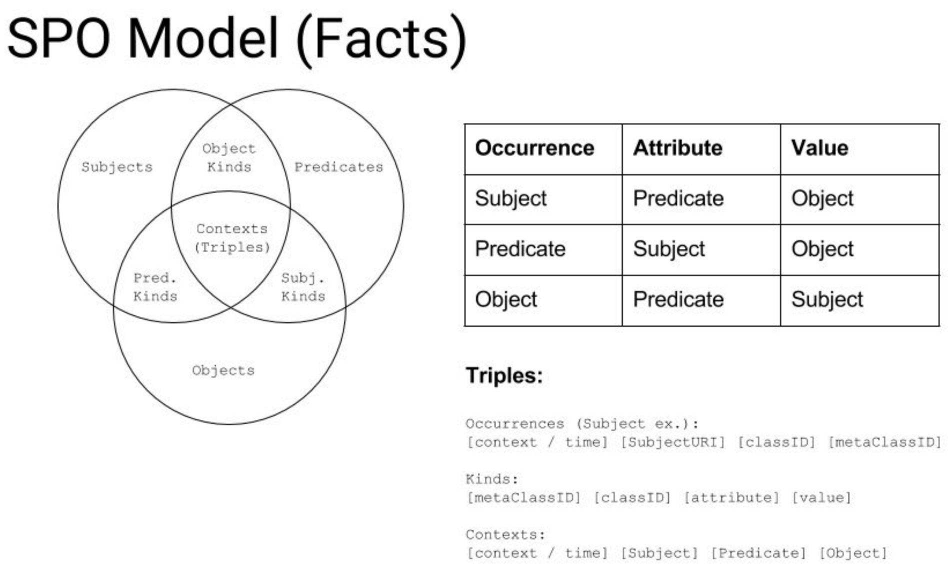
The second level: information (objects, signs and concepts) adds a ‘meaning’ layer to previous pure signs, treating them all (SPOs of the previous level) as a single statement triple (here named Topic) part, its Object. It also treats previous level Triples and Kinds as its Subject and Predicate, respectively, in its Topic triples (more on Kinds and how to use them below). This resembles more accurately what ‘semantic’ means in a semiotic context.

The third level: knowledge (behavior) aggregates the previous level in the same manner than information level did with data level. Roughly this level will correspond with semiotics branch ‘pragmatics’. The intention here is to infer as much as possible ‘state’ knowledge regarding the occurrence of events, the application of some rule or the conditions of some workflow.

Reactive dataflow. Aggregation.

TBD.

Statement Layers:



SPO Model (Facts)

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

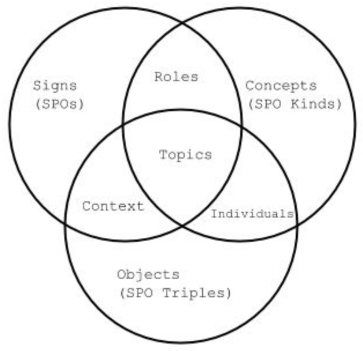
Triples:

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

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

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

Semiotic Model (SCO, Contexts)



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

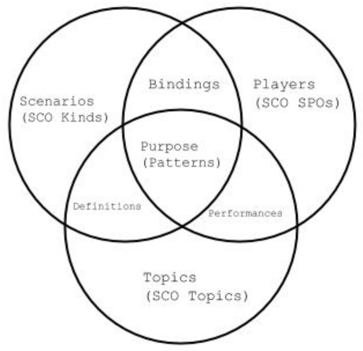
Triples:

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

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

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

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]
```

Alignement: ID. Merge

Metaclass - Class - Instance relationships expressed also as order comparison relations (example: a subclass has a superset of the attributes of its superclass, Resources defines other Resources by their aggregation contexts).

Alignment: Order, contexts

The comparison of resources in respect to a given Term (parent, child, previous, next, current, first, last, single) predicate in a given context (Term, Resource, Context, Resource) is meant to allow query based inference algorithms and also the (also dataflow enabled) integration of external machine learning / Big Data engines such as Google TensorFlow (given the correct encoding of resource IDs). Relations (image schema?): containment, part of, neighbor, cause of, etc.

Alignment: Attributes, Links

Example: (Ctx.: rel, Peter, Joe) : where neighbors, then friends, and then partners. Transitions. Truth values (temporal, for ordered 'names').

Relationship browse context example: 'causeOf'. Temporal contexts comparison using octal values. Services result Resources (Events). Selector Resource (pointers this, that, they), vars, wildcards, composite Resource, reference model results Resources.

Functional API

All hierarchy classes (including Functor and Term interfaces) are defined in terms of RDF Quads. Each Quad instance context aggregates other Quads instance contexts according their meaning.

Later we'll see how aggregation combines with dataflow activation for the query/traversal and inference over the resources graph.

Data input (Statements) activates (dataflow) on contexts (Definitions) and interactions (Declarations). Node infers / aligns / augments from input Statements. Definition / Declaration hierarchy: AST / Monadic parser combinators on input Statements. Node aggregation of Kinds as Statements (features).

Reify Model (Metamodel, TBD.).

The Quad statements are of the form:

Quad : (Player, Occurrence, Attribute, Value);

Functor (functional wrapper of Resource):
(Functor, Resource, Term, Functor);

Term (bound function / dataflow op wrapper of Resource):
(Term, Resource, Functor, Functor);

Resource<T> : Monad.
Parser: Resource hierarchy AST.
Parser Functors: consumes inputs.

Bound functions (Term : Resource) : query / traversal / transform.
Protocol APIs: (‘Someone’).fmap(‘Employee’) : List<Kind> SomeoneEmployments.

Resource Functor: Resource<T> hierarchy. Resource hier : AST. Parser Functors. Bound functions (Terms):

Example:

(‘Someone’ : Resource).flatMap(Employee : Kind) : EmploymentKind : Someone’s jobs.

Kind : (Person : Kind, Someone : Resource, Employee : Class, Employee : Kind);

Class : (Employee : Class, SomeoneEmployee : Resource, Employee : Kind, SomeoneEmploymentsKind : Kind);

TBD: Encode functional calls / activation graphs as quads (Resources in declarative transforms).

Functional features (Functors / Terms)

Parsing / Traversal / Transforms (functional pipes XSL: flatMap, CRUD paths, patterns / selectors). Selectors (Semantic patterns IDs, shapes / constraints, ‘positions’).

Materialize / augment order rels, comparisons. (Complement, Start, Finish).

TBD.

Nodes dataflow integration

Message / event driven augmentation.

TBD.

Application Demo

Dashboard (CRUD enabled) of aligned / merged syndicated data sources. Dynamic analysis / mining / drill generation. Inferred business use cases (domain driven development) frontend.

Dashboard. Admin. Features (integration). Domain translation. Alignment (merge) Actionable items (dataflow, activation, drill, mine, browse). Stubs other features.

TBD.

Implementation Details

- Functional expressions as Resources. Pattern IDs, variables, placeholders, dialogs (dataflow: encode graph).
- Actor / Role (class / metaclass). Context / Interaction. DCI reactive / dataflow. Promotion (event roles).
- Graph encoding. Naming scheme (Profile IO translation)
- Dimensional reified quads: (Dimension, Fact, Measure / Unit, Value);
- NLP: Substantive, Verb, Adjective (computer, computes, computed).
- Upper (internal) OWL / RDFS ontology. Restrictions based alignment.
- Infer equivalent properties by equivalent property domain / range (kinds). Domain / range kind alignment, ID, links, order inference by equivalent property kinds. Encode inferred schema as upper OWL / RDFS Statements.
- Reactive Streams: Subscriber, Publisher, Subscription, Processor <T> (Web, XSL Declarative Resource Dataflow Pipes).
- Resource: Data. Subscription(s): Context (actors / roles). Processor(s): Interactions (behavior instances).
- Discovery. Internal upper ontology. Sust, Verb, Adj. Reify layers entities. Terms (reified grammars, dimensional, rules). Constraints. Shapes. Domain / range, ordering rels comparisons inference. Schema merge / sync.

Appendix: Monads, DCI. Functional API

Monads:

- Parameterized type M<T>.
- Unit function (T -> M<T>).
- Bind function: M<T> bind T -> M<U> = M<U> (map / flatMap: bind & bind function argument returns a monad, map implemented on top of flatMap).
- Join: liftM2(list1, list2, function).
- Filter: Predicate.
- Sequence: Monad<Iterable<T>> sequence(Iterable<Monad<T>> monads).
- Order by contextual comparator of Definition Declaration.

DCI (Data, Context, Interaction):

TBD.

Functional API:

TBD.