Web of Data: P2P Business domain translation of problem spaces. Semantic Business Integration (WIP draft)

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

1 - Scope

Enable seamless integration of different datasources and services from different deployments and platforms to interoperate. A system A, for example, in the domain of CRM (Customer Relationship Management) and, a system B, in the domain of HMS (Healthcare Management System) would be able to be plugged into a 'bus'. Then, actions (CRUD, service invocations) in one system should have 'meaning' for the other (potentially many) systems.

2 - Metamodel

For enabling different sources of data and services behaviors to be treated in an uniform manner a fine grained metamodel of such artifacts should be done as for enable description and discovery of 'alignments' which are the means for one element understanding each other element behavior.

A Metamodel renders schema / instances of data, information and knowledge layers for a given domain.

The 'class' hierarchy, modelled as RDF quads has the following pattern:

Classes and their instances are modelled as a quad hierarchy of OOP classes:

ClassName : (instanceURI, occurrenceURI, attributeURI, valueURI);

A quad context URI (instance / player) identifies an instance statement (in an ontology) of a given OOP class. All ontology quads with the same context URI represent the same 'instance' which have different attributes with different values for a given occurrence.

An instance (player) may have many ‘occurrences’ (into different source statements / quads). For example: a resource into an statement, a kind into a fact, etc.

For whatever occurrences a player instance may have there will be a corresponding set of ‘attributes’ and ‘values’ pairs determining, for example, if the player is having a subject role in an statement then the attribute is the predicate and the value is the object of the given statement, the aggregated pairs of those occurrences, in common with other instances, the ‘subject kind’ of the resource, thus performing basic type inference.

Metamodel abstractions are wrapped into Functional Programming ‘monads’ (Resource) with reactive dataflow event driven semantics. Dataflow graph encodes dependencies / dependents for activation graphs tracking (input, state, output). Publisher, subscriber, subscriptions and exchanged follow a DCI (TBD) design pattern.

Distance: Dimensional metadata of Functional result Resource. Comparable (order).

Dimensional (semiotic) statements pattern: reify x is y of z in w statements. A statement will hold references to this dimensional data in their occurrences.

Upper ontology. Reified Metamodel plus encoding (IPs). Classes / Function / Kinds as Resources.

Grammars. Kinds Statement patterns.
Sustantivar (S); Verbalizar (P); Adjetivar (O); Pasión, Acción, Estado.

Upper ontology: exchange, merge, align. ISO 4D (object, time, place, attribute, value). Exchange of reified and instance metamodels (schema + instances). Vocabulary: semiotics sing + concept + language.

Normalize URIs hierarchy:

Normalize Metamodel hierarchy into Resource IDs aggregable statements.
Reify Metamodel hierarchy into Resource IDs aggregated statements.

Inferir / aggregate Kinds / Instances of compound URIs.

Example:

Topic: (Statement, URI, Metamodel);

Topic: Amor (Persona / Metamodel, Amante / URI, Pareja / Statement);

Reified Metamodel statements now could be aggregated for Resource ID inference.

Given that:

- Parent: superclass (occurrences).
- Attribute: Parent superclass (occurrences) attributes.
- Value: Parent superclass occurrences attributes values.

And the ‘monadic’ type which homologue all the hierarchy is of the form:

- Resource<Observable<T extends URI>> : T occurrences

The resulting ‘class’ hierarchy, given the following ‘notation’:

- Context (class) : Superclass (S, P, O);

is as follows:

- URI : (URI, URI, URI)

- URI (T): REST Semantics. HATEOAS / HAL, JSON-LD API. Reactive uniform representation API (HTTP verbs) through Resource monad.
- URI is the parent class of, for example, DBURI, RestURI, SPARQLURI, SoapURI, etc. which are the types that, wrapped by a monadic Resource, provides the connection mechanisms to the underlying protocols.

- Metamodel : URI (parent, attr, value);

A Metamodel is anything that could be modelled or represented by URI statements.

- Statement : Metamodel (parent, attr, value);

A Statement (reified) represents a Context Subject, Predicate, Object (CSPO) structure.

- Topic / Purpose : Statement (parent, attr, value);

A Topic is a set of Statements ‘talking’ about the same subject.

- Class : Topic (parent, attr, val);

A Class is the aggregated set of Topics ‘talking’ about the same kind of resources.

- Behavior : Class (parent, attr, val);

A Behavior represents an abstract ‘action’ which may be carried.

- Flow : Behavior (parent, attr, val);

A Flow is a concrete instance of a Behavior, for example a transition between Class attributes.

An aggregation algorithm is to be provided so it is possible to infer / aggregate topmost Flow(s) from original URI / Statement(s). Also, as Resource is a reactive designed monadic wrapper this algorithm performs in a publisher / subscriber manner and it is able to ‘react’ the underlying model given activation or inputs matching consumers which will propagate changes to its subscribers.

3 - Ontology Alignment

Datasource interoperability

Normalize input statements (URIs):

Translate any datasource to the following tabular roles:

Tabular roles: (Table, PK, Column, Value);

Statement roles: (Player, Occurrence, Attribute, Value);

Traversing quad elements:

Previous element in quad: metaclass.

Current element in quad: class.

Next element in quad: instance.

Aggregation: Same URIs parent / child of previous / next URIs.

Instance: current role URI / previous role aggregated URIs. Final: datatype / primitive. Example: age.

Kinds: current role same URIs / next role aggregated URIs. Example: Person. Final: Top (Object)

Alignment roles: (Datatype / Primitive, Kind / Instance, Kind / Instance, Kind / Instance).

From 2 datasources the final datatype / primitive 'aligns' sources statements (1 and 2 different sources to be merged):

Alignment roles 1: (Carpenter / Guiseppe, Grandparent / Giuseppe, Parent / Peter, Person / Joseph);

Alignment roles 2: (Carpenter2 / Giuseppe2, Grandparent2 / Guiuseppe2, Parent2 / Peter2, Person2 / Joseph2);

Context Kind / Context Instance: should be in both sources as the value of the player role of the statement. Metaclass of subject Kind.

Sliding (disambiguation):

Datatype / Primitives: regarding Context: Carpenter / Profession 'slides' to subject position, Bronx / Town occupies context (sliding till reach primitives).

Chained Kind (SPOs): Identify lowermost (top superclasses, objects) of Kind hierarchies. Aggregate Kind / sub Kind hierarchies. Sub Kinds: attributes are superset of those of super Kind.

Assign instance, class, metaclass and player roles. Player role: Datatype / Primitive or Kind / Instance.

If player role ends in Datatype / Primitive alignment could be performed.

If player role ends in Kind / Instance a new statement should be made 'sliding' player to class and recalculating hierarchies.

Match Datatype / Instance are the same values: Training. Mappings: learn hierarchies for known matching players. Semiotic alignment (sign is object for context in concept).

Semiotic Function: Languages: terms, semiotic mappings. Augmented dictionary (a is b in ctx for d).

Match Datatype / Instance are the same values: same Kind / Instance hierarchies taxonomy (graph shape). Abstract graph of models nodes (Kind roles: previous, current, next). Merge aggregated graphs augmenting Kind with attributes (Instance lattice). Attribute similarity function.

Alignments (merge):

- ID / Type merge: matching resources: same entities.
- Link / Attribute merge: matching resources sharing unknown attributes in context augmentation.
- Order / Context merge: matching resources with 'wildcards' in contexts / CSPO axes which respect Kind / Instance hierarchies. Wildcards will be provided with a comparator function.

Reifying Metamodel: Normalize Metamodel hierarchy into Resource IDs (Metamodel) aggregable statements.

4 - Algorithmic Alignment

Objective: encode URI statement quads into numeric quads (ResourceIDs) for, for example, Machine Learning algorithms or virtual network addressing (IPs) for resources.

The idea is being able to use a fixed length quad of fixed length components being able to 'reuse' a same numeric identifier with another meaning in another quad / component position. This could be used for virtual network addressing of statements for enabling protocols and operations.

As quads components are themselves metamodels (quads) this process could recursively encode a vast amount of information in a reduced fashion and enable, by the use of an identification algorithm, the use of known Machine Learning tools for processing large amounts of data.

Statement (C, S, P, O):
(meta:(super:(class:inst))

- inst: Primes sequence count (InstanceID)
- class: InstanceID product by corresponding InstanceID next primes (ClassID).
- super: ClassID product by corresponding InstanceID next primes (SuperID).
- meta: SuperID product by corresponding ClassID next primes (MetaID).
- Next prime is in relation with parent prime: parents aggregate child primes count.
- If element repeats, same prime is used as in first occurrence.

- Example
- (Carpenter, Grandfather, Father, Person);
- (FirstBorn, Brother, Son, Person);
- FirstBorn and Carpenter shares Person factor(s). Different identifiers in different contexts will preserve this characteristic.

- Reduce quad products disambiguating common factors.
- Primes may be mapped to their position in primes list: (1, 2), (2, 3), (3, 5), (4, 7), (5, 11), etc. Encode / decode from mappings for succinct representation.

Ontology Alignment from Algorithmic Alignment:

Reify Metamodel hierarchy into Resource IDs aggregated statements. Each Metamodel subclass entity has its corresponding (normalized) aggregable Metamodel statement.

Inferir / aggregate Kinds / Instances of compound URIs.

Example:

Topic: (Statement, URI, Metamodel);

Topic: Amor (Persona / Metamodel, Amante / URI, Pareja / Statement);

Reified Metamodel hierarchy statements now could be aggregated for Resource ID inference. Alignment with (upper) ontologies could be done using ID, Link, Context ordering functions.

4 - Functional Alignment

Monadic wrapping of hierarchy elements allow for the application of the above alignment mentioned methods by means of functional composition.

By means of monadic application of different alignment functions, Ontology Alignment and Algorithmic Alignment may be used, the first to perform the desired alignment operation on Resource(s) and the later for the comprobation of correctness in the reasoning / inference process.

Core alignment functions are: Identity alignment (to identify equivalent entities), Link and attribute augmentation and contextual ordering / sort function. Also CSPO 'getters' functions and RESTFul HATEOAS functions (by means of HAL or JSON-LD) should be provided.

Because Metamodel(s) aggregate knowledge in such a way, 'model' functions may exist which, for example, reifies the application of a Behavior to an entity instance (a Flow).

Semiotic Function: Languages: terms, semiotic mappings. Augmented dictionary (a is b in ctx for d). Grammars (ontology Kinds aggregation).

Verification of Ontology Alignments:

ID / Type align augmentation: ResourceID patterns. Applied Functions: Resource.fmap(AlignFunction<Resource, Resource>).fmap(MergeFunction<Resource, Resource>) : returns merged Resources.

Order in context align augmentation: ResourceIDs order in context / role (attribute, value axis). Applied Functions: Resource.fmap(OrderFunction<Resource, Resource>).fmap(ContextFunction<Resource, Resource>) : returns first.

Attribute / Link: align augmentation: common shared factors. Applied Functions: Resource.fmap(TypeFunction<Resource, Resource>).fmap(AttributeFunction<Resource, Resource>) : returns attributes.

Resource Functions: repository of functions (Dataflow specs, DSLs). Core Functions / declarative functions namespace for retrieving from exchanges and templates.

5 - Solution Architecture

The proposed solution architecture is the implementation of custom extensions to an ESB (Enterprise Service Bus) platform which allows for the deployment of custom URI(s) (DB, service, connector, etc.) implementations metamodel components.

By means of declarative 'templates' the platform will be allowed to specify and compose diverse source / service components (local or remote) enabling them to interact with each other. Interactions will implement the 'domain translation' pattern thus enabling gestures / behaviors in one domain to be translated into corresponding actions into other domains.

Declarative DCI Templates: Java / XML - XSL implementations (www.cactoos.org).

Layers (schema / instances):

Data
Information
Knowledge

Protocol:

Routes
Dialog
Activation Graphs (reactive)

Strategy patterns in functional composition.

Server less (distributed command message pattern).

Integration with documents / office platforms. Forms. Templates.

Monads: TBD.

Type constructor.
Unit.
Map / flatMap
Sequence (apply to list)
Chaining of mappings of functions.
Predicates.
Selectors (patterns).
Stream operations.

Functions:

Domain (Resource(s) URI hierarchy).
Namespaces (Java / Templates).

Resource(s) as functions: Inferred functions from domain. runtime merges function / Resource streams.

Activation

Dataflow

Functional ‘DOM’ (Java / Templates)

CRUD: Immutable Resource(s). Changes creates new entities in new contexts (axes).