Abstract Upper Ontology

The metamodels described are aligned into a common vocabularies ontology (RDFS/OWL, ISO 15926).

Aligned RDF/OWL model. Facades. ISO 15926. SoLiD Containers / Ports (Facades, Templates).

The basic attempt is to rely on standard Big (Linked) Data datasources (RDF, OWL, SPARQL and other formats/protocols endpoints), 'RDFized' datasources (relational or other endpoints/protocols via 'Adapters') then provide a set of features thanks to an 'internal' scheme of models and representations (not strictly RDF, described in the topics in this document) which will provide, in an ESB 'components' fashion the building blocks for 'declaratively' compose 'Ports' (endpoints/protocols/formats) which may or may not produce strictly standards based Semantic Web applications (other kinds of applications are allowed).

For this we use 'Facades' built upon 'Templates' (both described later) and one can, for example, expose an SPARQL or RDF(S)/OWL endpoint (aligned to the classes and properties of an ontology of choice) using a 'Protocol' implementation, expose an ontology through SOAP or perform a object-semantic mapping in some dynamic language in the form object-relational mapping performs for databases.

Features:
Type inference.
Attribute inference.
Link inference.
Merge (identity).
Sort (temporal / logical).
Alignement.
Dataflow via predicates.

Message flow (Template graphs)

Ports / Adapters are specific Container types.

Protocols <-> Container(Port/Adapter <-> Facades <-> Model layers).

Container layout: discover / wire bindings. Models, Grammars and Facades for a metamodel 'layer'. Templates encoded as Model's data. Apache Service mix / Sun's JXTA (DHTs) deployment (FCA, Protocol learning).

Architecture

Templates (addressable Resources, declarations, transforms). Messages. Pattern matching (Resource IDs). Dataflow. Example: addressable (Resource) request Template, create: POST, populate: Dataflow messages (from Template to Resources and from Resources to Template), retrieve state: GET. Resource creation via Templates (Containers, Models, SPOs, Kinds, Statements). Resources: Dataflow message reactive objects (Protocol).

Resources metamodel

Resource : Set (Predicate). Multiple URIs (identity), ID: URI in context? OWL Resource.

Predicate (Resources metamodel): Templates (expressions) Resource IDs selectors / patterns (identity: inclusion, order: inclusion depth).

Resource ID: S P O : (S.ResID, P, O). P/O also have similar Resource IDs.

A triples (quads) input graph is processed and aggregated into metamodel levels layers.

All metamodel levels: Facts, Objects, Purposes (layers) statements and helper resources are encoded into quads following the same pattern, from source SPOs to behavior (Purpose) quads.

An upper layer takes its base layer statements as subjects of their own statements. Then it takes base layer Kinds (types) as its predicates. Finally, base layer SPOs are layer's objects.

Each layer statements, types and individuals (models) are represented by the following metamodel classes. The rest of the schema is described by instances of the following classes.

Class Predicate.
- holds(resource : Resource)

Class Set.
- predicate : Predicate

Class Resource (extends Set).
- URI : String
- resourceId : Resource
- context : Resource
- subject : Resource
- predicate : Resource
- object : Resource
- parents : Map<Resource, Resource>

- roles : Map<Resource, Resource>
- mappings : Map<Resource, List<Resource>>
- previous(ctx : Resource, state : Resource) : List<Resource>
- next(ctx : Resource, state : Resource) : List<Resource>
- children(ctx : Resource) : List<Resource>
- add(res : Resource, role : Resource)
- list(role : Resource) : List<Resource>
- apply(template : Resource) : Resource
- owlRdfResource : Node

Resource dataflow: Reified Resource ID as CSPO, callbacks, events, messages, Templates. Add resource (res : Resource, role : predicate/set). Children collection derived from parent. Update.

add(res : Resource, role : Resource)
list(role : Resource)
apply(ctx : Resource, res : Resource)

Protocol: Template exchange, for required state, KB asks for necessary modifications until everything is asserted. Protocol learning: classification / regression in the form of Template exchange / population (FCA).

Statements, Kinds, CSPOs, Resources

Each metamodel level has its own meanings for their statement components. The most important helper resource derived from statements are Kinds. Kinds are a form of type inference based on a resource attributes and values. A given SubjectKind, for example, is a 'kind of' Subject.

A Subject has, given this model, some parent statements for which its role is Subject (resource model instance) and plays a SubjectKind.

A SubjectKind aggregates Subjects sharing the same set of Predicates in different statements inferring their 'class'. It then can aggregate this Kind with statements with the same attributes (Predicate) with those with the same values (Object) inferring their 'metaclass'.

The same accounts for PredicateKinds and ObjectKinds with their respective attributes and values. TBD.

Resources metamodel levels

The different metamodel layers, aggregated as stated before, are the following:

The Facts layer attempts to arrange input statements and their components into sets: Facts (actual statements), Subject, Predicate and Object sets and their respective Kinds. It also manages statement contexts. TBD.

The Objects (semiotic) layers arranges previous (Facts) layer SPOs into 'Signs', Kinds into 'Concepts' and statements into 'Objects'. Its statements (Topics) are of the form:

(ctx : Topic) (object) (concept) (sign)

Its Kinds are: TBD.

The Behavior (purpose) layer arranges previous (Objects) layer OCSs into 'Players', OCS Kinds into 'Roles' and statements (Topics). Its statements (Purposes) are of the form:

(ctx : Purpose) (topic) (role) (player)

Its Kinds are: TBD.

Facades are resources that group related resources (Templates). They may be considered as another model layer. For example all Facts for a given subject, all order Objects, all order management purposes. Their statements are of the form:

(ctx) (Purpose) (Object) (Fact)

Resources model instances

For each model level a similar structure of sets of statement, kinds, and SPOs is represented by actual instances of resources 'modelling' each layer entity types. Each type represents a similar 'role' in each layer. For example, Facts, Objects and Purposes are all kinds of statements as objects, signs and players are all objects (role) in statements.

Contexts, Models: Facts, Objects, Purposes.
Statements: Fact, Topic, Purpose.
Subject, Object, Topic.
Predicate, Concept, Role
Object, Sign, Actor.

Same (aggregated) instances may be shared across metamodels. Each resource (set) definition will match appropriate instances.

Resources functional mappings

Each resource has resolution of the entities corresponding itself in respect to other resources or contexts:

(Triple, Kind) : SPOs
(Triple, SPO) : Kind
(Kind, SPO) : Triple
(S, O) : P
(P, O) : S
(S, P) : O
(SK, OK) : PK
(PK, OK) : SK
(SK, PK) : OK

ID Res: common superset, common rels / links.

Data: SPO / Resources (reifying types & behavior).

Types: OCS / Kinds.

Behavior: TSP / Statements (grammars).

Resources reference model

Reference model attempts to provide an uniform manner of accessing resources metamodel and applying transformations (Templates).

This is mainly achieved by sorting hierarchicaly and horizontally a graph of resources in specific contexts.

Methods of resource are: parent, children, previous and next. Example. TBD.

anEmp.next(Emp, Dev) : Leader.

aTask.parent(aProject) : aContract.

Kinds operations (reification, hierarchies)

Kinds (types) may be reifyied as their corresponding SPOs as means to establish type hierarchies or as a metadata facility for augmenting their meaning.

Also, using reifyied Kinds there could be expressions matching resources with a given Kind, common supertypes or common links and relations.

Resource IDs

Given a resource, this concept aims to populate an 'ID' augmenting it with all resource's occurrences and the Kinds in that occurrences context.

Then, given a proper encoding, a resource 'ID' may serve to match or find similar (or equivalent) resources and query information between them. Also and expression can be built to retrieve a desired pattern of resources.

Resource ID: TBD.

Resource ID (Subject): (S.resourceID, P, O) for all statements of S.

Primitive terms.
Opposites.
Negation.
Inverse.
Complements.

Term has term in role in context. Classes.

Terminal has terminal as non-terminal in production/rule.

non-terminal has non-terminal as terminal in production/statement.

Resource ID: Template. Terminals: SPOs, non-terminal: Kinds. Statements, ordered by statement context:

(context, role) (rule, non-terminal) (statement, terminal)

Context / Role (classes) calculated through aggregated term identifiers.

Resource ID statements context: temporal metadata. Reify events (resources), temporal relations (compare) other resources/events. Truth values.

Resource ID 'relationships' selectors (ordered) for: Peter, Joe: Neighbors, Friends, Partners. For Peter, Developer: Peter's Position (reified, order comparable). For Trainee, Senior: Transitions. Truth values (temporal).

Encode 'mask' node location in hierarchy. Query / self arrangement: parents, this, children (mult. occurrences). Infer tree / mappings (Resource IDs, reference model). Reactive (dataflow) Templates / evaluation time resolution.

Resolution: inference, learning, networks, models (representation specifications).

Actions (Model layer Templates / Predicates): previous, current, next (Possible in context / order, hierarchy / mappings reactive resolution: reference model). Materialized (events, messages) from DCI (Deployment features)

Events, Flows, Rules

Event: SK. Prev. TVal. Passion. (flow, rule).
Flow: OK. Act. TVal. State. (state, rule).
Rule: PK. Next TVal. Action. (state, flow).

Identity / Order metadata: Predicates (inclusion, depth). Metadata: reifyied Context. Example: Ctx. S: act, P: prev, O: next.

Definitions / Instances. TBD.

Grammars

Grammars are models built from actual models Kinds, using Kinds as SPO, respectively, in statements ('rules'). For a grammar:

Kind: non terminal
SPOs: terminal
Statements (of kinds): production rules

Lambdas: Types (Kinds / OCSs), Data(SPOs), Behavior (Grammar / Statement, TSPs).

Data, information, knowledge. Information classifies data (data: price, information: price variation) in some given axis / context. Knowledge aggregates information into behaviors (some variation, tendency).

Monads: wrappers, lift operations / types into domain (int + int : int, int + string : string).

Reify Temporal, Reference, Mappings relations. Measurement. Units, Dimensions: birth(Y2000) : 2000's births.

Templates / Protocol

A Template is a set (graph resource) of statements with any layer SPOs, Kinds, Statements, variables and wildcards. It also may contain reference model (hierarchical) or functional (mappings) expressions. A Template may give form to a Facade and be the means of interaction with resources apply(ing) them as transformations.

Also, identity resolution (align and merge) and ordering (temporal and arrangement) can be expressed in term of Templates. Resource IDs and Template resources are the basis for dataflow implementation

Once submitted to a resource (apply) a Template starts a dialog in a 'protocol' with request / response cycles in which each part asks / replies till no resolutions left.

Predefined Templates: Metamodel Predicates, Inference / Learning, Facade aggregation Templates.

Alignment: Adapter / Port Templates.

Application Templates (Protocol declarative messages)

Dataflow via Predicates: Templates, Events, Flows, Rules. Callbacks.

Example: TBD. Messages / apply (Containers, successive contexts).

Facades

Facades are models narrowed to an specific context in an specific level. Facades for Facts regards to one particular Subject (anOrder) Facades for Objects regards to a particular Topic (orders) and Facades for Behavior regards to one particular Purpose (orderManagement).

Facades: models with TSP, OCS, SPO statements. Dataflow. Resource.apply(Resource).

Identity resolution / merge

Use grammar and mappings to create all possible statements between graphs to merge. Calculate Resource IDs. Merge matching IDs resources. Resolve ambiguity using functional mappings and reference model.

Identity resolution: all possible statements, all possible Resource IDs. Merge with actual resources, refine possibilities with Template dialog (contexts, this, that, theirs 'pointing' variables) mappings / reference model.

Contexts: reified contexts (identity, ordering). Truth values (statements, resources) in contexts. Reifyied ordering (events, flows, rules types/instances) temporal/contextual 'holds' values (octal). Merge Template: tautology. Dictionary: sameAs.org. Term in context (actor, role, topic).

lang(concept) : sign. object, topic (ctx).

Alignement: Ps (column) equivalent: shares Os (values). Os equivalent: shares Ss. Ss (PK column) equivalent: shares Ps. TBD.

Mappings relations (predicates): assert (materialize) inverse, opposite, reflexive, transitive & symmetric predicates statements (ie: something weight 200, 200 weightOf somethings*). Materialize RDF(S) / OWL inferences.

Attributes and links discovey

(dept) (leader) (peter)
(joe) (worksAt) (dept)
: (joe) (boss) (peter)
TBD.

Streams. Learning, analysis, mining, CEP, rules, flows, inference, augmentation. Templates with reference model and functional mappings operations. Dataflow (eval Predicates).

Ordering and temporal alignment

A series of statements like this may be presented:

(someone) (wash) (car)
(someone) (takesOut) (car)
(someone) (takesIn) (car)

Infer correct ordering. Encode ordering kinds: (actual) (prev) (next). Operate over ordering: encode position in three bits. TBD.

Order inference: opposites, complements (truth values, contexts). Order 'kinds': actual, prev, next (Subject temp context SPOs). Events, Flows, Rules (verb action, passion, state). Bounds.

Dimensions, axes, units, measures.

Primitive terms.
Opposites.
Negation.
Inverse.
Complements.

SK / OK: Complements. Ordering. Statements reification into SPOs.

PK Inverse: swap S/O.

Opposites: P respect to PK, three possible values: original value, opposite, negation. Expression. TBD.

take out/wash car complement of wash/take in car. take out/take in opposites. Opposites: order SPO. Predicate: statement with P of both kinds. Determine S (first) by PK state/ordering (Resource IDs). TBD.

Types (Kinds) 'natural' ordering (Dimensions, Units). TBD.

Applications

Abstract representation metamodel (alignment into an interoperable upper ontology(ies)).

BI (Business Intelligence) and Big Linked Data. Learning: regression, classification. Dimensional analysis, Units.

Sources (Adapters) and Endpoints (Ports) for various protocols. Dashboard, management UX for deployment customization. Browser (engine). Drive / Docs 'gestures' (Template driven).

Wrappers for existing applications services / endpoints (SOA, JMS, JDBC) augmenting / enhancing interactions in contexts.

Enhanced applications. Abstract declaratively representations of DCI. Augmentation. Big Linked Data. Hooks into wrappers.

Dashboard / Management
UX: Facades (higher, lower sliding metamodel levels, Facts, Objects, Behavior model types relative to evaluation position) hierarchical aggregation: Profiles, bindings. Facade (Profile) occurrence in more than one level / context. Roles, relations (as part / container). Contextual browse and faceted search. Build, materialize Templates. Configure bindings. Configure Port / Adapter endpoints (Profiles, Templates. Import / export (Protocol, Profiles, Templates).

Deployment features

Distributed declarative discovery and remote binding of container entities. Messaging based container / models interaction (Templates, Protocol).

Containers layout: TBD.

Models layout: TBD.

Container (uniform modelling, may play different roles / bindings):
Models: SPO.
Grammars: OCS.

Facades: TSP.

Container, Models / Grammars, Facades: onMessage(msg : Template). Resolution: Message route applies to Container, Model, Facade.

Message oriented (Templates):
Containers - Models / Grammars - Facades hierarchy.

Message dispatch (broadcast) to Containers, Models and Facades. Resolution. Each level populates (response) it’s part of the Template with its corresponding part of knowledge. Partially populated response (Template) traverses again the message hierarchy or gets re-posted along containers (broadcast).

Alignment: Identity between Facades & Resources. TBD.

Deployment: everything is a Model (Template: reactive Model): Statements, SPOs, Kinds, Metamodels (levels), Containers, Facades, Adapters, Ports, Profiles. Messages / Events through Templates. Publish as (micro) Services. Platform, resolution (OSGi/JXTA). Container wraps model for publishing as a service (MetamodelService, FacadeService, ProfileService: one for each Model instance) SPO, Kind, Statement services auxiliar to previous services (scoped instances). Naming, Index and Registry services 'global' (models profile resolution).

Templates: messaging / events dispatch. Routes establishment source-dest (mult.) Template dialog (Protocol) scopes. Profiles. Model Lookup registry, naming, index (hierarchical parent-this-child refinements). Identifiers: payload and Model endpoint(s) (Resources). Payload Resource (Template) is materialized as a Model (see DCI interactions/models) instance (context interaction model / use case dialog instance) reifyable and addressable.

Persistence of dialog state (Templates, events/messages) reifies materialized Model's state. Contexts, interactions and temporal metadata. State / behavior bindings. Traceability of Resource state regarding interactions (Templates I/O and relations due to Resources definitions or references).

Empleado (inst: child de this en parent) porque tuvo rol (this) en contratación (ctx: parent).

Adapters

Container with models translated from some datasource. Port Adapters. Events (messages) updates / update from source models.

Ports

Container (Models, Grammars, Facades) exposed in layers transformed from Models into an specific protocol representation (REST, SOAP, OData, RDF/OWL, SPARQL, Facades: Microservices).

Lab

Storage as processing. Addresses as operations / operands. Values as results / addresses / subsequent operations (apply Resource). Values Resource metamodel encoded as a single quad (rest of model as graph operations) Transmission as storage (buffers). Mappings of functions, Reference model, patterns, Templates. Discrete. Composition for indeterminate domains / ranges (dimensions, units, radix). Octal (temporal) values. Templates. Patterns (Resource IDs). Network addresses. Image addresses. Addresses: quads IDs (data and operation) (op(data(op(data)))). Dataflow: update values data / ops.

Implement memory (storage) based controller: low level metamodels representations (C, JNI). Emulated (interface).

Implement message driven metamodels (Architecture: Resource, Templates) over controller. DHT translation (peers, controllers). Java, JXTA.

Implement nodes (Facades) uniform interfaces of underlying representations (Protocol, Ports). Composable as models / layers (bindings). JXTA, NodeJS.

Facades 'generated' (inferred / aggregated) dynamically. Ports stated declaratively (Templates).

Facades. Microservices. Profiles. Highly specialized. Profile discovery: index, registry, directory. Interchangeable Facades. Metamodel levels hierarchical Profile IDs (matching entities, objects, behaviors in parent-this-children iterative / sliding manner). Agents: models that discover models (Facades / Profiles) according its own metamodels and bind them (reciprocally). P2P / JXTA.

DCI: Interfaces, behavior / interactions declaration. Classes, sets (hierarchy) membership, data model. Kinds: behavior / model bindings (model state result from interactions). Actions models (Resource IDs).

Datatypes: ie. PredicateKind: SKs / OKs (Recursive, Grammars, primitives / enumerable types / by Predicates / categories, concepts). Characterization encoding (FCA) by factors, power values, exponential or trigonometric functions. Resource IDs: Templates (reactive dataflow) ie: president of Argentina (bidirectional).

Datatypes: ie. PredicateKind: SKs / OKs (Recursive, Grammars, primitives / enumerable types: dentro, fuera, antes, hoy, mañana, dar, recibir, esto, eso, aquello, etc. Align compositions by contexts / dialog: term roles / by Predicates / categories, concepts). Characterization encoding (FCA) by factors, power values, exponential or trigonometric functions. Resource IDs: Templates (reactive dataflow) ie: president of Argentina (bidirectional).