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

© 2018 Sebastian Samaruga (ssamarug@gmail.com)

Introduction

Functional / Reactive abstractions.

Keywords:

Semantic Web, RDF, OWL, Big Data, Big Linked Data, Dataflow, Reactive programming, Functional Programming, Event Driven, Message Driven, ESB, EAI, Inference, Reasoning.

For enabling different sources of data and services interfaces behaviors to be treated in an uniform manner a fine grained functional metamodel of such artifacts is to be built / aggregated as for enabling description and discovery of functional merge, aggregation and 'alignments' which are the means for one element understanding each other element data, schema and behavior (for example meaning of identity, attributes and contexts).

The goal is to streamline and augment with analysis and knowledge discovery capabilities enhanced declarative and reactive event driven process flows of applications data and services between frameworks, protocols and tools via Semantic Web backed integration and Big (linked) Data applications enhancements. Perform EAI / Semantics driven Business Integration (BI).

Provide diverse information schema merge and syndicated data sources and services interoperability (for example different domains or applications databases). Translate behavior in one domain context into corresponding behavior(s) in other context or domains via aggregation of domain data into knowledge facts.

Aggregate knowledge into component metamodels which can interact between each other (databases, business domain descriptions, services, etc) into a dialog-enabled protocol via dataflow / activation semantics which leverages referrer / referring contexts with knowledge enabled from participating models.

Enable seamless integration of different data sources 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) plugged systems.

Basically the idea is to functionally 'homogenize' data sources: their data, schemas and (inferred) 'behaviors' (flows / transforms / processes). By means of semantic aggregation into layers of Metamodels and three 'alignment' models of sources (identity of records / entities without common keys via class / metaclass abstractions, resolution of missing relations / attributes and contextual 'sorting': for example cause / effect in a process events context) one should obtain the following layers (alignment / backend metamodel):

1. An homogeneous (functional) metamodel of data and schema (Resources). Like an XML document encoding Metamodel aggregated data / schema and behavior from plain RDF input. Aggregation alignment / Resources Metamodel.

2. An homogeneous (functional) metamodel of entailed 'Templates' (transforms / flows) and the behaviors they entails (integration of, for example, action / flow A in origin X entails action / flow B in origin Y). Resembles an XSL transform aggregating previous Metamodel Resources into Kinds. Ontology alignment / Templates Metamodel.

3. An homogeneous (functional) metamodel of entailed 'Queries'. Resembles an XSL transform over previous Templates, aggregating their Kinds as Resources as the possible behaviors (queries) an application might have. Algorithmic alignment / Queries metamodel.

The joke resides in that those metamodel layers, being obtained, aggregated and aligned from raw data (an RDF dump of a database, for example) are to be parsed functionally being an upper model ‘code’ for the previous model ‘data’.

So, aggregate and align diverse data sources in homogeneous structures which provides Queries which apply over Templates (code / flow / transforms) which applies over Resources (data) where they have determinate structure or 'shapes' (Metamodel messages resolution time).

Document Web vs. Data Web.

The current Web (and the applications built upon it) are inherently ‘document based’ applications in which state change occurs via the navigation of hyperlinks. Besides some state transitions on the server side by means of ‘application’ servers, not much has changed since the web was just an ‘human friendly’ frontend of diverse (linked) resources for visual representation.

Even ‘meta’ protocols implemented over HTTP (REST) are layers of indirection over this same paradigm. At the end we are all ending up spitting HTML in some form or another. And much of this seems like a workaround over another while we still trying to get some juice using ‘documents’ for building ‘applications’.

The Web of data is not going to change this. And it’s not going to be widely adopted because the only thing it has in common with ‘traditional’ Web is the link(ed) part. No one will ever figure out how to build Web pages with ‘Semantic’ Web. It’s like trying to build websites with CSV or XML files: they are just data exchange formats. That’s not the role in which SW will shine, it’s another set of format for which there are many protocols.

Semantic Web is a set of representation (serialization) formats and a bunch of (meta) protocols which excels for the modeling and accessing of graphs. Graphs of… well, graphs of anything (anything which may have an URI, at least). Let’s call a graph a set of nodes and a set of arcs between nodes, these are Resources. A triple is a set of three Resources: a Subject Resource, a Predicate Resource and an Object Resource (SPO: node, arc, node). A Triple may have eventually a fourth Resource, a Context, then the Triple (Quad) has the form: CSPO.

Now imagine a CSV or XML file (or a database engine SQL dump) that given this input files could relate it with a lot of other files, ‘calculates’ missing fields or figures out new ones. It may also figure out new ‘rows’. This is what Semantic Web has of ‘semantic’ and exactly what it has not of ‘Web’, in the traditional sense.

So, SW is a data exchange mechanism with formats and protocols. For user agent consumption it has to be rendered into some kind of document, like it is done for any other (graph) database. But the real power of using the SW approach is for machine consumption. We’ll be using it that way in our example approach of EAI / Business Integration as a metamodel encoding facility which will entail aggregation and reasoning of business domains facts and flows. We’ll be using ‘ontologies’. An ‘ontology’ is for SW format representations as a database schema is for queries / statements only that this schema is modelled as SW resources as well.

Inputs:

Syndicated datasources, backends of diverse applications databases, services interfaces (REST / SOAP / JMS, for example) should be aggregated and merged (matching equivalent records, for example) via the application of ‘Semantic’ metamodels thus providing via virtualization and syncing interoperability between those applications.

Features:

Once consolidated metamodels of the domains involved into the business integration process are available, services metamodels come into play providing alignment (ontology matching, augmentation and sorting), transformations and endpoint features.

Connectors: The goal, once source data / services are consolidated and aligned is to provide APIs for different languages / platforms which enable consumers of those data / services retrieve rich ‘augmented’ and enhanced knowledge that was not present in the original (integrated) backends via ‘connectors’.

Goals:

Given applications (big or microservices) there should be a way, given its schemas, data and behavior to ‘infer’ (align into a semantic upper ontology) what this applications do (semantically speaking).

Given a business domain problem space (data, schema and behavior for an application, for example) a ‘translation’ could be made between other domain(s) problem spaces which encompasses a given set of data, schema and behavior instances (objectives) to be solved in the other domains in respect of the problems in the first domain.

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 for being able to work and interact with them via some of the following features:

Feature: Data backends / services virtualization (federation / syndication / synchronization). Merge of source data and services.

Any datasources (backends / services) entities and schema regarded as being meaningful for a business domain translation and integration use case, regardless of their source format or protocol.

Diverse domain application data with diverse backend databases and services and diverse sources of business data (linked data ontologies, customers, product and suppliers among others) are to be aligned by merging matching entities and schema, once syndication and synchronization are available.

Examples: different names for the same entity, entity class or entity attribute. Type inference.

Identity alignment: merge equivalent entities / instances.

Attributes / Links alignment: resolution of (missing / new) attributes or links. Relationship type promotion.

Order (contextual sorting) alignment: given some context (temporal, causal, etc.) resolves order relations (comparisons).

Infer business domain process semantics and operations / behavior from schema and data (and services). Aggregate descriptors (data, events, rules, flows) into Metamodel messages.

As mentioned before an existing deployed application could benefit from this framework integrating its data and services when plugged into the bus. It then may be enhanced using actual flows to consume augmented knowledge or being available to / for consumption by other applications or services via ‘connectors’.

Example

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 an institution) as an increase of money amount dedicated to flu prevention or treatment. In the media or advertising domain the objectives of informing population about flu prevention and related campaigns may be raised. And in the technology sector the tasks of analyzing and summarizing statistical data for better campaigns must be done.

Abstract Metamodels

Metamodels: aggregated encoding of data, schema and behavior (metadata). Messaging scheme (Nodes).

Aggregated backend for message exchange and Node layers backend models. Unified interfaces for functional dataflow. Kinds aggregation (layers).

A Metamodel Resource is a (functional) monadic type which wraps a reference of its quad’s context resource URI (resource of which it has occurrences into quad’s subjects) and a (dynamic) list of its occurrences (parent / children).

TBD: Functional DOM. Monad interface (Type), Bound Function interface (Member), Application interface (Type / Members instances declaration). Functional Application / Binding Statements: (Application, Monad, Function, Monad); Bound functions. DCI.

URI : Endpoint. Representation. REST URIs activation API. Uniform resource interfaces (DOM, HATEOAS HAL / JSON-LD).

Kinds: For a given statement, for example. the ‘Subject Kind’ of its Subject will be the aggregation of all Predicate / Object pairs of its occurrences (as Subject in other statements) performing basic type inference. Example: Person for some subject(s) who has ‘name / value’ and ‘age / value’ into their occurrences. The same applies for Predicate and Object in statements.

Kinds: For whatever occurrences an URI may have into an statement (for example being the subject of one statement) there will be a corresponding set of ‘attributes’ and ‘values’ pairs (predicate and object, respectively, in this case) determining that the aggregated pairs of those occurrences, in common with other instances conforms the ‘subject kind’ of the resource, thus performing basic type inference. There may also be ‘predicate kinds’ and ‘object kinds’ with their corresponding SPO attribute / value pairs.

Abstract Metamodel Classes:

The ‘class’ hierarchy of a Metamodel, modelled as RDF quads has the following pattern:

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

ClassName : (instance URI, occurrence URI, attribute URI, value URI);

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 different occurrences.

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.

The ‘monadic’ type which homologue Resource hierarchies is of the form:

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

T : URI. REST Semantics. HATEOAS / HAL, JSON-LD functional (verbs) uniform API. Representation. Activation. URI classes: DBURI / SOAPURI, etc.

Class descriptions:

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

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

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

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

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

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) and the layers between. 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.

Metamodel hierarchies (Resource / Template / Queries):

Statement: (Metamodel occurrence):
(Statement, URI, URI, URI);

Metamodel: (Topic occurrence):
(Metamodel, Statement, URI, URI);

Topic: (Class occurrence):
(Topic, Metamodel, Statement, URI);

Class: (Behavior occurrence):
(Class, Topic, Metamodel, Statement);

Behavior: (Flow occurrence):
(Behavior, Class, Topic, Metamodel);

Flow: (Flow, Behavior, Class, Topic);

Functional Metamodel messages activation protocol (CRUD / Message / Verbs): TBD.

Monads: everything as a Resource (of Observable of aggregated T). Unify treatment of data coming from any datasource into streams of aggregated layers from source data.

Activation (query / match). Monadic transforms.

Reified Metamodels (upper ontology templates). Primitives, roles. Bitmap coding. Quad encoding. FCA. TBD.

Alignment / Metamodels stack

All three Alignment layers must provide aggregation, identity, links, attributes and contextual sort Alignments, performed in three Node 'layers' for three Metamodel backends. Metamodel messages resolution resolved by each layer controllers, models and views activation.

Layer Node(s) configures Alignment controllers and Metamodel backends for Metamodel messages resolution / message views. Aggregation alignment relies on Resource Metamodel backend. Ontology alignment relies on Template Metamodel backend and Algorithmic alignment relies on Query Metamodel backend.

A Metamodel renders schema / behavior / instances of data, information and knowledge layers for a given domain (connector). Metamodel layers (resource, template and queries) are 'inferred' by Node instances via Metamodel message exchange resolution.

Metamodel layers (stack): aggregated Metamodel stack over previous Metamodel Resource Kind(s) as Resources of this Metamodel. Models of stack of Alignment methods. Semantics of aggregated knowledge (resources, templates, queries) Kind layers.

Aggregation, Alignment, Inference / Entailment.

Models:

Resource Metamodel: aggregates plain statements from Resource till flows.

Template Metamodel: aggregates Kinds (grammar model: special type of Resource Monad).

Query Metamodel: Templates / Resources layers monads (functional dialog: variables, wildcards, placeholders). Alignment / Resolution.

Alignment models:

Aggregation

Ontology

Algorithmic

- Metaclass: classes this class is attribute of.
- Class: attributes.
- Instance: attribute values.

Resolution time (message exchange).

TBD: Resolution services: Index (from identity alignment, query by 'content' keys model matching identities), Naming (from links / relations alignment, query types: classes / metaclasses applicable to model), Registry (from contextual order relation alignment query for model position in context). Query Metamodel resolution (query encoded as Resources).

Metadata / Models. Inferred functions from domain. Functional 'DOM' (Dynamic Object Model / Type Object). Java / XSL / JavaScript transforms.

‘CRUD’ operations inserts new Resources (axes, contexts): functional CRUD API for Metamodel (streams).

Functional application. Activation.

Aggregation Alignment: Resources Metamodel

INCOMPLETE.

Layers plugged into Runtime Nodes message dispatch. Aggregates plain Resources.

Objective: Context alignment (resolve x is y of z in w); IO / Activation / Reasoning.

Encoding (occurrences: context, player, attribute, value): S(P, O); P(S, O); O(S, P); C(S, P, O); Shapes / grammars (kinds).

Resolution: (name, player); (context, attribute); (index, value); IO / Activation / Reasoning.

TBD: Dimensional aggregation. X is Y of Z in W (contexts / dimensional statements modelling). Versions, contexts, scopes (access control). Attribute / links rel / scope (referrer context).

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

TBD: Tabular aggregation. Table, PK, Column, Value quads input.

TBD: Dimensional aggregation. X is Y of Z in W (contexts / dimensional statements modelling). Versions, contexts, scopes (access control). Attribute / links rel / scope (referrer context). Distance.

Resources: CAM / Functionally encoded mappings: Aggregation alignment. Metamodel (URIs).

Ontological Alignment: Template Metamodel

INCOMPLETE.

Layers plugged into Runtime Nodes message dispatch. Aggregates Resources into Kind grammars.

Objective: Merge (align / blend) graphs with equivalent ‘records’ without common keys algorithm. Identity resolution / merge.

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.

Ontology alignment: (Metaclass, Class, Instance, Occurrence);

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

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

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

From 2 datasources the final (object) 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, Sex / Male 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.

Update:

Ontology align: aggregate inputs

(class / instance / metaclass, class / instance / metaclass, class / instance / metaclass, class / instance / metaclass);

Context: objects hierarchy; Object: primitives; classes / metaclasses: kinds; Identity merge: equivalent objects / primitives (kinds) shapes / paths. From equivalent primitives traversal (slides) to equivalent objects.

Class: Kind relative to Resource attributes / values.

Metaclass: Kind relative to Resource (class) occurrences as value of attribute of other Resource(s).

Ontology example:
(class / "Computer Supplies" / metaclass, class / "Notebooks" / metaclass, class / "Lenovo" / metaclass, class / "Lenovo XYZ" / metaclass);

Ontology example (slide left):
(class / "Notebooks" / metaclass, class / "Lenovo" / metaclass, class / "Lenovo XYZ" / metaclass, class / "Lenovo XYZ S/N 1234" / metaclass);

Ontology example (slide right):
(class / "Inventory" / metaclass, class / "Computer Supplies" / metaclass, class / "Notebooks" / metaclass, class / "Lenovo" / metaclass);

Templates: Ontology alignment over Aggregation alignment. Metamodel: Grammars (Kinds) and Resources.

Algorithmic Alignment: Query Metamodel

INCOMPLETE.

Layers plugged into Runtime Nodes message dispatch. Aggregates Template Kinds into behavior queries.

Objective: Succinct representation of quads for graph encoding / exchange and validation of ontology alignment.

Objective: infer links / rels missing or calculated through ontology.

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.

Queries: Algorithmic alignment over Ontology alignment. Metamodel: Kinds (grammars), Resources.Functional dialog: variables, wildcards, placeholders (via encoding).

Resolution: Messages

Functional reactive protocol (dialog) for Runtime Nodes / Layers. Streams (async). Topics, routes, scopes between Nodes (bus).

Resolution time (message activation between Nodes): perform Aggregation, Alignment, Parsing, Transforms, etc. (model layers, transforms, facade, connector, runtime reactive dataflow interfaces).

Node: publisher / consumer of Metamodel encoded message streams. Controller ‘resolves’ messages: model ‘activates’ on message contents (CRUD), view publishes ‘activated’ message results.

Resolution: Metamodel message over three alignment methods: identity, attributes / links, contextual sort. Perform Metamodel parse results (Resource / Templates / Queries) alignment.

Metamodel transforms (over Query, Template, Resources parse). Queries: aggregated possible behaviors / 'purposes'. Parsed with Templates at resolution time. Then parse Resources. Render possible 'operations' (purposes) translated to / from (metamodel layers resolution / ‘invocation’ translation) aggregated Metamodel resolutions / facade 'views' (declarative functional reactive DCI / connectors specific 'ports': service, database, etc. from / to translated layers quads). Discoverable 'purposes' (from facades, peer orchestration).

Alignment: populate / sync / aggregate Metamodels (Resources, Template / Grammars, Queries / Behaviors) at Metamodel message resolution time.

TBD: Templates (Resource merge): Parser combinators / apply Resource as Function (transform). Facades. Functional Interfaces (client / server 'peers').

CUD (CRUD / ABM) / versioning of ‘records’: Contextual order relation alignment to, for example, a temporal event or Resource(s). Provenance. Resolution.

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

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). Example: ‘raiseSalary’ flow.

A repository of such functions may exist, beside some core Functions / declarative functions namespace for retrieving from exchanges and resolution (Functions embedded into Metamodel messages reified as Resources).

Messages:

Normalized message format which may encode / activate behavior in any route(s) layer.

Entire hierarchy encoded (Resources / Templates / Queries). Message streams (from Metamodels): Query / Template / Resource streams. Parsing: apply message from Query to Template to Resource (i.e.: Behavior application over a Metamodel results in an Statement stream).

Resolution Aggregation:

In the context of a resolution exchange of dataflow messages aggregation of Metamodel layers is performed.

Resolution Parse:

In the context of a resolution exchange of dataflow messages parsing (functional composition of Queries, Templates and Resources) of Metamodel hierarchies is performed.

Resolution Alignment:

In the context of a resolution exchange of dataflow messages alignment of Metamodel layers by algorithms is performed.

Messaging: Messages flow through dynamic routes between Component Metamodels and Components. Synchronization and propagation of knowledge by means of activation.

Message: Determines next routes / payloads for its Component Metamodels routes / payloads. Aggregates activated knowledge (routes / payloads) in its exchanges (browsing / discovery agent). Relative to its ‘referrer’ (Context: ‘Developer’, ‘Peter’, ‘Work’ -> ‘Peter Working as a Developer’).

Message: Metamodel instance (Data routes facade, Information and Knowledge payload body / template). Activation / aggregation determines Message facade state for further routing / payloads. Goal: fulfill specific template.

Domain Flows (scenario / context, roles, state, transitions). Ontology aligned Messages. DCI: Data (Metamodel, Message), Context (sharing scopes, subscriptions, roles inferred from data / referrer), Interactions (declaratively stated in routes APIs).

Classes / RDF literals / Primitives (parse type kind assignment: Address, Age, etc.) Alignment algorithms.

UX: Grammars. Possible domain / range / properties. Dialog alignment.

Functional / Reactive programming. Resolution interfaces: domain / range input / results (scope, contexts, levels, content / activation). Index (identity, ‘content’), Naming (links / rels, IDs), Registry (context positions).

Activation / REST / Pipelines (P2P, addressing, message self routing, Resolution). Templates. Provenance. Addressable 'transmissions' (exchanges, listeners). Materialized transforms / operations / routes (cache / patterns / shapes / grammars).

Shape expressions (upper ontology, grammars, facade, activation). LDP. SoLiD. REST HATEOAS (HAL, JSON-LD) Resource(s) addressing / interface generation (Resolution interfaces).

Nodes, Runtime, Peer interfaces

Node interfaces: streams / endpoints (controller, model, view). Layers transforms performs aggregated model layers functional parsing (query over template over resources). Alignment resolves resulting parse entities augmentation.

Node, Runtime, Peer message dispatch hierarchy. Configuration: Node interfaces implementing classes. RT Nodes: resolves corresponding Metamodel messages through activation.

Node(s) configuration (injection of implementing classes):

(AlignmentCtrlImpl, MetamomelModelImpl) : LayerViewImpl;
(ResolutionCtrlImpl, LayerViewImpl) : TransformViewImpl;
(FacadeCtrlImpl, TransformViewImpl) : EndpointViewImpl;
(ConnectorCtrlImpl, EndpointViewImpl) : AdapterViewImpl;
(RuntimeCtrlImpl, AdapterViewImpl) : PeerViewImpl;

Actor / Role model: Functional DCI interfaces for message exchange between Peer runtime components (Nodes).

Data: Metamodel messages.
Context: Node (resolution) roles.
Interaction: Activation over message streams.

Runtime: Alignment / Metamodels aggregation (resource, template, queries). Input connectors. Queries are Templates. Templates are Resources (aggregated / parsed / materialized as).

Metamodels (Resource / Templates / Queries) are wrapped Resources into a functional stream of (hierarchical) Messages which provide the means for consumption / production of knowledge.

TBD: Protocols: Message interchange are wrapped into resolution exchanges. Routes / Endpoint activation. Dialogs (reactive resolution of patterns / primitives: variables, wildcards, placeholders in streams).

Templates (domains / dataflow subscriptions / contexts). Templates metamodels (domain / purposes) dataflow translation of domain behaviors.

Resolution time parsers:

Monadic parser combinators apply built from upper layers (Queries parse Templates which parse Resources Metamodels) as Functions (transform / merge). Build combinators for each Metamodel hierarchy layer that accepts Query (behavior),Template (context / schema) / Resource (data) as arguments.

Queries generated from domain Templates Kinds knowledge aggregation (behavior / interactions inference).

Templates generated from domain Resources Kinds knowledge aggregation (schema / roles inference).

Combinators: apply layers domain transforms: query, templates, resources. Perform alignment and aggregation on parse results (populate models) resolution.

Parser monads: Metamodel layers (of Resource of layer type) parsers. Parser result AST: Metamodel (Parsers of Resource hierarchy, combinators into Parser of Metamodel). Templates generated from domain Resources knowledge aggregation (schema / roles / behavior inference).

Resource Metamodel: equivalent to XML document.

Template Metamodel: equivalent to XSL templates for Resource Metamodel.

Queries Metamodel: equivalent to XSL templates for Template Metamodel.

Runtime resolution: aggregate / align Resources Metamodel (Metamodel Message layers streams). Data.

Runtime resolution: aggregate / align Templates Metamodel (Metamodel Message layers streams). Schema.

Runtime resolution: aggregate / align Queried Metamodel (Metamodel Message layers streams). Behavior.

Parsing mechanism:

0. Parsers of Metamodel layers. Inputs: (context : Template layer, data : Resource layer).

0. Parser matches Queries layers.

1. Parser matches Query Templates layers.

2. Parser matches Template Resources layers.

TBD: Templates (Resource merge): Parser combinators, apply layer Resource as Function (transform).

Metamodel resources (Resources / Templates / Queries) follows a reactive Publisher / Subscriber pattern as Endpoints. Runtime configuration. Streams. Dataflow. Messages.

TBD: Routes are determined by aggregation and knowledge domains (protocols / scopes / contexts). Message dispatch. Transforms. Node activation.

Resource API: Activation. Reactive. Dataflow. Controller, models and views of configured Node(s).

Metamodel message representation interfaces: Browseable hypermedia (REST HATEOAS HAL / JSON-LD).

Dialog Protocol:

Scopes. Wildcards. Variables. Placeholders. QA request / response.

Dataflow / Runtime transform hierarchies (Node's configuration / streams):

Runtime resolution exchange streams (messages: protocol / routes via Observables): Queries, Templates, Resources. Lambdas.

Dataflow / Runtime transforms (exchanges / apply merge):

Session / Dialog API. Facade / APIs (Class / Behavior / Flows).

Apply aggregation. Apply alignment Perform parsing. Populate models.

Lambdas example:

```
Template(cook);
Resource(Ingredients):
Template(serve);

Resource(ingredients);
Template(cook):
Resource(meal);
```

In the context of a data dialog given matching Information a knowledge template could be matched which activates a Rule Flow (pattern / transform) which updates players LHS with RHS.

Purpose Metamodel (Queries): Task accomplishment services / QA. Over Domain Metamodels. Over Backend Metamodels.

Purpose Metamodel (Queries): aggregation ontology, layer scopes (facts, concepts, roles / contexts : Data IO, dialog state, behavior templates).

Layer scopes for Purpose Metamodels:

Purpose Metamodel (Templates): Connector populates behavior templates layer from dialog state from previous facts and other Metamodels. IO activates dialog state to / from behavior templates and aggregation augments roles / contexts. IO parses / renders facts aggregated to / from dialog state in respect to behavior templates contexts.

Purpose facts (Resources) Connector: Connection IO (render hierarchical flows, prompts, confirmations from dialog state relative to current inputs) into facts to / from dialog state in respect to behavior templates (NLP Connector).

Templates (aggregated knowledge) determines what to prompt for input and what output facts apply for the current dialog (Information) 'session' (Data). Other Metamodels augments and get augmented from these interactions (retrieve database records, invoke services, alignment 'interprets' user input) thus integrating QA into a broader set of integration use cases.

Current input fact in respect to dialog context resolves next behavior template to be populated into dialog context (question / question, question / answer).

Current output fact in respect to dialog context resolves to expected behavior template(s) to be populated into dialog context (answer / question).

Dialog state session mediates between facts and knowledge in question / answer scenarios (hierarchical flows, prompts, confirmations).

Purpose Metamodel Connector (NLP, parser / renderer) handles representations of dialog facts 'questions' and 'answers' in the context of available question / answer sets (behavior templates knowledge) in the context of a dialog session.

Facade, Peer, Adapter, Endpoint

A special type of Node configuration classes are Peer, Adapter and Endpoint (and their controllers, models and views). Their message resolution mechanisms rely on the implementation of custom IO modeling. DB, services, etc. pattern adapters and interactions via encoded Metamodel messages from other layers.

Upper ontology (interface) adapters. Node interfaces providing for a Peer’s Runtime configuration via resolution of Metamodel message exchanges. This layers of Nodes deal with a highly aggregated representation of all ‘plugged’ domains (sources) of knowledge: data, schema and behavior aligned from integrated backends.

Facades. Interfaces (functional declarative queries: schema / data / behavior). Metamodel encoded queries. Queries API translation by Peers alignment. Resolution services.

Client (Facade) Models: APIs. Transforms from Facade interface queries (Client Metamodels).

Object model (type object / upper ontology). APIs. Facades. Grammars.

APIs:

Facades: Metamodels / Models.

Metamodel / Templates. Endpoints (publish / subscribe layers Messages). Transforms (exchanges):

Interface Endpoint (Node) (publisher / subscriber: protocols Peer, Node, Connector, Facade, etc. methods / instance, context, scope).

Interface Message (RDF / aggregated Metamodel / scope, context). Template / Metamodel hierarchy propagation (correlations).

Interface Exchange (Templates / protocol scope / context Metamodel transforms / routes, publish).

Runtime: nodes configuration, activatable message flows, data, context (subscriptions), interactions.

Dynamic upper ontology:

If models shared by Peers share some common underlying ‘upper’ ontology, this could be aggregated, aligned and matched by only Metamodel data itself. That means no ‘hard-coded’ upper ontology but one that is dynamically shared and exported to other formats if necessary.

Upper ontology: Aggregate Metamodel(s) Metamodels. Peer Node level. Reify Metamodel hierarchy into Metamodel statements for each class. 'Dynamic': alignment methods between Peer(s). Align to OWL / RDFS models (SPARQL / endpoint).

Upper purpose ontology: Modelled in Metamodel (reification). Constraints. Restrictions. Temporal / dimensional parts. ISO like ‘templates’, OWL, RDFS Metamodel IO export / import (endpoint).

Metamodel Object Model (endpoints / connectors APIs). Metamodel facade: Object graph / RDF APIs / others. Upper ontology aligned. Grammars (dialog alignment). Facade messaging / activation / utils.

Appendix: Adapter example. RDBMS Connector relational (draft)

Datasource connectors for providing the means of sync / translation between backends and Node(s) Metamodel message exchange resolution. RDBMS example.

Input patterns (relational adapter: RDBMS connector triple producer):

Category / Sub Category (metaclass / class / instance).

Master / Detail.

Item / Item instance.

Item / Status (enum).

Status / Actions.

Action / Response (status flows).

Dimensional (measure, item, dimension, value).

Proposal: de-normalize all the schema in one table. First column: table name. Second column: PK / ID. Apply ML, reducers and aggregation for each case. Emit corresponding quads.

Input patterns (service adapter: SOAP / REST connector triple producer): TBD.

Input patterns (linked data adapter: LDP / SPARQL connector triple producer): TBD.

Other adapters (services: REST, SOAP, etc.).

(TBD).