## Brochure

Machine Learning and Big Data for the Enterprise

Augment existing backends (connectors).  
  
Expose augmented backends (extensions).  
  
Distributed reactive / streaming message driven architecture (dataflow).  
  
CRUD / flows applications: predictive models. Inference for entity completion / process fulfilment.  
  
Document applications: classification / tagging.  
  
Collaboration / communication applications: roles / state (actions) in interaction contexts.  
  
Domains. Integration. QA: 'wizard' like application frontends.  
  
Data Intelligence.  
  
Cloud based service infraestructure. No deployment or installation required.  
  
Description:  
  
Could it be possible that existing applications (backends, datasources / services) as they are deployed today could benefit from semantic tools alignments, augmentation and learning (inferences) enabled by a framework leveraging a 'virtualization overlay' which covers not only 'intelligence' in a given application domain but between applications. This domain 'intelligence' being 'encoded' as domain behavior knowledge allowing to describe 'alignments' ('translations') between such applications.  
  
The point is if it is possible maybe, to some degree, 'automate' this domain (application: backends, datasources and services) virtualization, integration and translations via the use of some form of heuristics, inference and learning enabling them to be augmented themselves and in respect to other applications domains with learning and knowledge capabilities in the most transparent manner.  
  
Those applications should be 'plugged' in a streaming bus (Nodes). Adapters (Backend streams / IO) allowing streaming adapter IO synchronization (domains 'gestures' translation) performing corresponding domain's 'effects' given context's actions.  
  
Features (enabling previously mentioned capabilities): aggregation (alignment: identity merge, augmentation: attribute / rels discovery, regression: entity 'role' in context discovery) by means of an uniform messaging layer and declarative 'assets' (components) described through an uniform Metamodel layer (Semantic Resources Metamodel REST APIs).  
  
Client Nodes: ad-hoc application extension assets (dimensional entities, schema, flows declarative descriptions) as means to augment bus applications with new knowledge. Custom declarative endpoints that expose APIs through protocols.  
  
Dashboard: virtualized domains visualization and assets management (domains use cases flow management).  
  
Example: dimensional data / schema / behavior flows in one application / domain generate 'triggered' transactions between applications / domains (CRUD + rules / flows application kinds). Infer backend 'contexts' (DCI / Metamodel).  
  
Example: classification (document oriented application kinds). Flows (trays: state / form action templates) classify images / documents (folders / labels) by features. Automatic tagging (labels).  
  
Example: query custom extension endpoints (protocols: inference / predictions). Apply 'views' transforms over aggregated bus domains. Expose knowledge in custom protocols (REST, SOAP, SPARQL, etc.). Complete missing information.

Could it be possible that existing applications (backends, datasources / services) as they are deployed today could benefit from semantic tools alignments, augmentation and learning (inferences) enabled by a framework leveraging a 'virtualization overlay' which covers not only 'intelligence' in a given application domain but between applications. This domain 'intelligence' being 'encoded' as domain behavior knowledge allowing to describe 'alignments' between their applications.  
  
The point is if it is possible maybe, to some degree, 'automate' this domain (application: backends, datasources and services) virtualization and integration via the use of some form of heuristics, inference and learning enabling them to be augmented themselves and in respect to other applications domains with learning and knowledge capabilities in the most transparent manner.  
  
Those applications should be 'plugged' in a streaming bus (Nodes). Adapters (IO backend streams). Streaming adapter IO sync (domains 'gestures' translation)  
  
Features (enabling previously mentioned capabilities): aggregation (alignment: identity merge, augmentation: attribute / rels discovery, regression: entity 'role' in context discovery). Assets. Peer Metamodel. Network. Domains 'translation'. Resources Metamodel REST APIs.  
  
Client Nodes: ad-hoc application extension assets (entities, schema, flows) augments bus applications with new functionalities. Custom declarative endpoints expose APIs through protocols.  
  
Dashboard: virtualized domain visualization and assets management (domains use cases flow management).  
  
Example: data / schema / behavior flows in one application / domain generate 'triggered' transactions between applications / domains (CRUD + rules / flows application kind). Infer backend 'contexts' (DCI).  
  
Example: classify documents applications (flows / trays: state / form action templates, classify images: folders by features). Automatic tagging (labels).  
  
Example: query custom endpoints (protocols: inference / predictions). Apply 'views' transforms over aggregated bus domains. Expose knowledge in custom protocols (REST, SOAP, SPARQL, etc.). Complete missing information.

## Introduction

Could it be possible that existing applications (backends, datasources / services) as they are deployed today could benefit from semantic tools alignments, augmentation and learning (inferences) enabled by a framework leveraging a 'virtualization overlay' which covers not only 'intelligence' in a given application domain but between applications. This domain 'intelligence' being 'encoded' as domain behavior knowledge allowing to describe 'alignments' ('translations') between such applications.  
  
The point is if it is possible maybe, to some degree, 'automate' this domain (application: backends, datasources and services) virtualization, integration and translations via the use of some form of heuristics, inference and learning enabling them to be augmented themselves and in respect to other applications domains with learning and knowledge capabilities in the most transparent manner.  
  
Those applications should be 'plugged' in a streaming bus (Nodes). Adapters (Backend streams / IO) allowing streaming adapter IO synchronization (domains 'gestures' translation) performing corresponding domain's 'effects' given context's actions.  
  
Features (enabling previously mentioned capabilities): aggregation (alignment: identity merge, augmentation: attribute / rels discovery, regression: entity 'role' in context discovery) by means of an uniform messaging layer and declarative 'assets' (components) described through an uniform Metamodel layer (Semantic Resources Metamodel REST APIs).  
  
Client Nodes: ad-hoc application extension assets (entities, schema, flows declarative descriptions) as means to augment bus applications with new functionalities. Custom declarative endpoints that expose APIs through protocols.  
  
Dashboard: virtualized domains visualization and assets management (domains use cases flow management).  
  
Example: data / schema / behavior flows in one application / domain generate 'triggered' transactions between applications / domains (CRUD + rules / flows application kinds). Infer backend 'contexts' (DCI / Metamodel).  
  
Example: classification (document oriented application kinds). Flows (trays: state / form action templates) classify images / documents (folders / labels) by features. Automatic tagging (labels).  
  
Example: query custom endpoints (protocols: inference / predictions). Apply 'views' transforms over aggregated bus domains. Expose knowledge in custom protocols (REST, SOAP, SPARQL, etc.). Complete missing information.

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 data sources, 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 data sources / 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 data sources (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’.

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.

## Metamodel

Kind, class: Data roles in context interactions / promotion (types). DCI / dimensions, data, schema, behavior:

The attempt is in trying to adapt an object-oriented design pattern (DCI: Data, Context and Interactions) to the realm of RDF Quads metamodels. There should be regard of dimensionally aggregated data (D) for which schema occurrences (C) play roles in behaviors instances / flows (I).

Resource (functional abstraction).

Resource Monad : Observable.

Resource Monad : application / domain bound functions. Templates. Selectors, Predicates, etc.

Metamodel : Resource (deployment runtime handles together with Peer Metamodel).

URI encoded entity / occurrence statements. Reification.

Metamodels: aggregated encoding of dimensional data, schema and behavior (metadata).

Messaging scheme (Peers):

Aggregated backend for message exchange and Node layers backend models. Unified interfaces for functional dataflow.

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. Alignments.

Message: Metamodel occurrence (render Metamodel / Peers into layers : TBD). Message: reified RDF graph / payload. Declarative Messages: possible messages in one domain:

Message: data, schema, behavior in context of one interaction.

(Metamodel, Message, Dimension, Occurrence);

Routing by Dimension / Occurrence alignments. Payload (context) by reified attribute / value. Verb (template) by resources activation (declarative reified graph).

Resource: de-referenceable Message representation (entity / occurrences).

Metamodel Layers (Dimension, Data, Schema, Behavior).

Metaclass: roles / occurrences (context)

Class: attributes

Instance: values

Reification (resources hierarchy): each layer statement is reified into its immediate upper layer (instance / class). Grammars.

API: Metamodel / Peer. Patterns resolution. CAM.

Dimensional roles: embeddings (operate / apply functionally over resources: bound functions).

Pattern:

(Context, Occurrence, Attribute, Value);

Layers (metaclass / class / instance):

Dimensional layer:

(Dimension, Unit, Measure, Value);

(Occurrence, Dimension, Unit, Measure);

Data layer:

(Entity, Occurrence, Dimension, Unit);

(Statement, Entity, Occurrence, Dimension);

Schema layer:

(Kind, Statement, Entity, Occurrence);

(Class, Kind, Statement, Entity);

Behavior layer:

(Behavior, Class, Kind, Statement);

(Flow, Behavior, Class, Kind);

Aggregation, Alignment, Activation: encoding / algorithms (example: ontology matching, attributes clustering, role identification).

Functional Resource. Resource types. Reactive (activation).

Data, Information, Knowledge helper bound functions (resource templates).

Event driven alignments: layers 'listen’ to messages layers class data.

C, S, P, O. Statements. Kinds. Context, SPO attributes / values (metaclass, class, instance). Aggregation, layers.

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 (predicates) with different values (objects) for different occurrences (statement subject).

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 (SP for an Object, SO for a Predicate).

Sets representation. Aggregation flows. Encodings (IDs, ops, embeddings). Sets encoding: joins / operators / results masks. Comparison results (temporal / set operations) octal values.

Sets encoding: signed octal digit. Comparisons / order rels.

Grammars: augmented metamodels. Templates. Message syntax (placeholders, variables, wildcards). Syntax layer over reified resources hierarchy (i.e.: kind / class resources). Parser (scope / context activation resolution).

CRUD: Functional (dimensional 'versioning’).

Upper ontologies / alignments (activation).

Semiotic encoding. Lattices / FCA. X is Y of Z in W (contexts / dimensional statements modelling).

(Context, Sign, Concept, Object);

Domains. DOM (Type, Content, Command) JAF. DCI.

C, S, P, O, Kinds (reified), Statements (reified). Layers. Aggregation.

Temporal / dimensional 'tags' for statement / resources. Translation (dimensional) statement resources in other dimensional occurrences (truth values). Aggregation. Alignments.

Reified Resources (Metamodel / alignments) for framework dimensional, data, schema and behavior layers entities.

Events. Messages. Protocols. Routing / Addressing: I/O. Alignment / templates.

Templates / grammars: contexts / reified Message Resources. FCA. NLP. ANN. Embeddings. DCI.

Resolution: alignments (index / identity, attributes / registry, roles / naming).

Metamodel Functional Resources: context groups occurrences, occurrences groups attributes, attributes group values. Methods (get, set / ctx, res).

Layers Streams (contexts). Wrappers: contexts, occurrences, attributes, values (methods).

Dimensional layers, alignment & reified hierarchies: augmentation (by classification, attributes and roles aggregation).

Resources encoding: class (Context URIs) 'whole life individuals' (entities).

Resources encoding: instances (Occurrences: Resource URIs quad statement subjects) w./ attribute predicates / value objects 'occurring' in determinate dimensional states.

Resource templates: Resource layer in / for which this Resource is an occurrence (example: Kind for an Statement). Functions (bound): Aggregation plus alignment. Parse / syntax (context). TBD.

Metamodel Resource Context, Occurrence, Attribute, Value. Resource (URI) / Statement  
Metamodel hierarchy layer encodes temporal alignments / dimensional features. Hierarchy layers 'inherits' corresponding materialized upper layers.

Metamodel Resources base layers instances (grouping) of materialized upper layers: behaviors schema, schema data. Behavior metaclass of data. Schema occurrence of behavior. Data occurrence of schema.

Temporal / dimensional contexts: resources / occurrences (URI level resource occurrence: dimensional context (attributes / values), occurrence as resource encoding  
dimensional rels. Dimensional Resources layer (class / instance).

Materialized dimensional attributes / values.

Reification. Upper alignment. ISO. Factors (ternary rels, embeddings).

Aggregation: prev / next layers CSPO / Kinds / Statements: (C: Kinds, S: Statement, P: SPO, O: SPO);

● Data (Schema, Behavior)  
● Schema (Data, Behavior)  
● Behavior (Data, Schema)

Semiotics: Syntax (grammar / schema: sign), Semantics (sign / object, data: object), Pragmatics (behavior: concept). Logic (inference / activation, networks / statements: embeddings / operations on terms encodings). Data, schema, behavior embeddings from semiotic encoding features: (Context, Sign, Concept, Object).

Syntax: operate on terms (Resources). Embeddings. Example: Kind(URI) / Statements. Others (addition, subtraction, contexts). Templates (wildcards, variables, placeholders).  
Feature extraction. Functional operations.

Ontology alignment: domain Peers translation, IO, backends / services.

Metamodel Functional Resources: context groups occurrences, occurrences groups attributes, attributes group values. Methods (get, set / ctx, res). Bound functions.

Layers Streams (contexts). Wrappers: contexts, occurrences, attributes, values (bound functions, domain templates).

Dimensional data, schema, behavior DCI. Metaclass, class, instances occurrences / contexts (templates).

Data, Information, Knowledge: dimensions / layers. Values (prices), relations (price variations), inference (price tendency).

Messaging: topics (dataflow bus) message broker. Routes (shapes / signatures: Endpoints, placeholders). Interactions.

Layers / Nodes: Dataflow Peer / Node layers (stateless bus nodes / layers specs). Declarative layer query / transforms. Contexts.

REST: Ontology backends / IO. Metamodel instances. Data.

Augmentation: Aggregation. Alignment. Activation.

Activation: message streams based inference materialization / alignments completion (reasonings). Aggregation, alignment dimensional data, schema, behavior (knowledge).

## Architecture

Description: Metamodel / Message driven / enabled nodes. Exchange in / out streaming queues over node topics. Consumes input / produces output when node acknowledges proper inference.

Description: Declarative statement / discovery of nodes: domains (IO backends) connectors, endpoints (IO extensions), processors (bus augmentations).

Description: nodes performing aggregations, activations and alignments over uniform message bus (broadcast / dynamic routes) exchanges.

Description: Metamodel / Message statements (RDF quads) encoding: aggregations, activations and alignments (augmentations / IO) discovery / routing / activation algorithms.

Description: activation / alignment (ontology) enabled declarative node behavior descriptions.

Description: exchange layers (scopes): domain, context, message (addressing, routes).

Message driven layers (Peer / domain) based on REST (Resource) abstractions. Stream, reactive (subscription based) abstractions. Metamodel based endpoints resolution. All REST abstractions (Resource, Content type, Representation, Verb, etc.) represented as resources (Metamodel URIs, i.e.: for Message, Endpoint, Exchange / interaction). Runtime (layers, declarative templates, IO).

Processing Layers (Node, Endpoint, Exchange, Event) : metaclass / class / instances (layers) corresponding to Metamodel resource layers. Each layer 'activates’ over subscription streams (discovery / message driven) of Peer (runtime Metamodel) messages with respect to its corresponding layer.

Peer (application / runtime ontology, alignments) Metamodel and domains Metamodel exposed via deployment runtime. Layers 'template’ expanded an 'executed’ in message contexts for declarative component description spec plus instance data.

Publish / subscribe of layer context declarative layout. Aggregation of messages and triggers (onMessage / onExchange: map / reduce).

Processing layer resolution (IO) of messages according their schemas (alignments / activation).

Request resolution: layers 'arguments’ in scope resolves DCI flow according layer template declaration:

Route(Message) : Context

Context(Resource) : Interaction

Interaction(Content type) : Resource

Streams: Resources (identifiers, types, representations). Dialog. Scopes.

Event driven alignments: layers 'listen’ to messages layers class data leveraging behaviors declaratively stated (i.e.: IO / CRUD operations when alignment of messages activates such behaviors).

Messages: Metamodel representations. Domain inference: discover domains 'assets’ by means of aggregation, alignment and activation. Render assets materialized into metamodel 'templates’. Augment (grammars).

Node

Endpoint

Exchange

Event

Activation, Aggregation, Alignments.

Layers publish / subscribe (streams, messages). Resolution queues (activation).

Messages encoding: Resources (layers) plus grammars (reification) syntax.

Inputs / Node sync: CSPO Message (layers) statement normalized from diverse backends (relational, dimensional, semiotic, etc.).

LINQ / Parsing: Metamodel and processing layers runtime contexts.

Abstract identifiers: resources and resource occurrences, attributes and values. Encoding. Paths. Patterns. Discovery, functional APIs.

Events. Messages. Protocols. Routing / Addressing: I/O. Alignment / templates.

Uniform model 'language' for framework APIs and application instances (domains). Declarative 'templates’ (metamodel encoded): layer 'assets’ (aggregation, alignment, activation).

Reified dimensional data, schema and behavior alignments rendering / activation APIs  
(contexts) with domains instances.

Reified Resources (Metamodel / alignments) for framework dimensional, data, schema and behavior layers entities.

DataFlow layers: publishes lower layers / subscribes upper layers entity kinds. Scopes, parser (streams events).

Responsibilities, lifecycle (scopes). Discovery: subscriptions description (address, signatures, payload, behaviors).

Node (resources metamodel, alignments / IO).

Node, Endpoint (Node resources addressing).

Endpoint, Message (Endpoint resource encoding / IO payload).

Message, Exchange (addressable interaction / context).

Exchange, Streams (Node resources 'topic').

Streams, Events (templates / 'filtering' activations / verbs).

DataFlow: Node, Endpoint, Message, Exchange, Stream, Event, Stream, Exchange, Message, Endpoint, Node.

Dataflow processing layers / Metamodel (Message encoded / alignment activation) layers correspondence (Peer layer activates on Metamodel layer. Node(Dimensions) : Endpoints, Endpoint(Data) : Exchanges, etc.).

Node / Dimension, Occurrence. Domain.

Endpoint / URI, Statement. Data.

Message, Exchange / Kind (Role), Class (DCI contexts). Schema.

Stream, Event / Flow (Verb), Behavior (Action).

Messaging layers functional behavior: Dataflow layers uniform declarative IO / behavior (reactive / streams activation between layers) specs encoded as Resources / Metamodels: declarative dataflow graphs (layers contexts). Templates (grammars / transforms / monadic wrappers: resource 'application' as rels, operators, embeddings) for system / domains metamodels. Aggregation, alignments (metamodel / domains parser: inferred / aggregated domain assets).

Alignment: A: index / search (identity, using B, C). B: registry / classes (attributes, using A, B). C: naming / roles (contexts, using A, B).

Ternary (recursive) functional API. Event sourcing (temporal reasoning / streams).

Set encoding (octal encoding / masks known, unknown facts order / comparison): temporal dimension (quads).

Temporal relations. Events (clock). Comparisons. Temporal encoding / operations (three octal temporal 'positions': past, now, future) holding resources in statements (CSPOs  
holding ordered resources).

CSPOs Resources encoded with functional temporal relations, example: now(past, future), given context statement.

Ternary (recursive) functional API. Event sourcing (temporal reasoning / streams).

Set encoding (octal encoding / masks known, unknown facts order / comparison): temporal dimension (quads).

Temporal relations. Events (clock). Comparisons. Temporal encoding / operations (three octal temporal 'positions': past, now, future) holding resources in statements (CSPOs  
holding ordered resources).

Dimensional data, schema, behavior DCI. Metaclass, class, instances occurrences / contexts (templates).

Data, Information, Knowledge: dimensions / layers. Values (prices), relations (price variations), inference (price tendency).

Messaging: topics (dataflow bus) message broker. Routes (shapes / signatures: Endpoints, placeholders). Interactions.

Layers / Nodes: Dataflow Peer / Node layers (stateless bus nodes / layers specs). Declarative layer query / transforms. Contexts.

REST: Ontology backends / IO. Metamodel instances. Data.

Augmentation: Aggregation. Alignment. Activation.

Activation: message streams based inference materialization / alignments completion (reasonings). Aggregation, alignment dimensional data, schema, behavior (knowledge).

## Deployment

Runtime: Container / Peer (domain) Metamodel. Messaging URI based 'bus’ (Peer / Metamodel Container instance resolving peers routes).

Monads / FP.

Contexts (DCI): Layers (model, processing).

Resources: Identifiers, Types, Representations.

Actors / Roles.

Blueprints: Stream subscriptions, discovery services (Peer instance, Metamodel resolution), layer templates. Domain Peer declarative Node / Metamodel setup.

Concrete identifiers: resources and resource occurrences, attributes and values. Encoding. Paths. Patterns. Discovery, functional APIs.

Sync. Messaging. Services (listener streams).

Containers. Services. Endpoints. Components (Metamodel, Resources). ServiceMix (OSGi), Spring, REST, RxJava / Jersey (CDI).

Streams. Functional concepts: Metamodel / Resources. DCI, Rx / Monads.

REST / DCI / HAL: Resources, Paths, Content Types (URIs), Accept (URIs), contexts, Representations / messages: data, Verbs (URIs, commands, context templates / message bindings): interactions (browseable methods / contexts). Referrer.

Message: aligned to source / dest activation schema, translated by backends plugins alignments. Example DB plugin: (create, altaEmpleado / entity, empleado / empStmt, URIupdates) : behavior / schema / data class / instance statements. Service / inference  
plugins aligns to corresponding classes.

REST Messages: syntax / templates. Query metamodel data, schema, behavior class / instance metadata (HAL). Verb semantics over message contents (template). Example: query Node endpoints by Dimensional contexts, Endpoint Exchanges by Schema contexts, etc.

REST / Reactive Layers:

Resources (Endpoint: request).

Representation (Messages, Content type: layers alignment).

Interaction (Streams, Exchange: contexts, Events: response).

Aggregation, Alignment, Activation.

Route: context flows, referrer. Dynamic routes: template / context.

Peer / container API: callbacks from hosted Node metamodels (activation / IO Providers). Event / ontology driven.

Plain inputs (activation quad messages statements) translated when activated (dataflow) to requesting plugins. Prev example message translated from plain input to backend plugin: builds messages.

REST Messages: syntax / templates. Query metamodel dimensional data, schema, behavior class / instance metadata (HAL / HATEOAS DCI). Verb semantics over message contents (template).

Functional application. Resource wrappers. Results flows.

Infer / aggregate / merge behavior, schema, data class / instances from backends.

Sources. Nodes (declarative backends: datasources / services).

Features. Assets (inferences). Activated merged domains knowledge (REST alignment / activation APIs).

Clients. APIs (declarative DCI contexts). Node role (expose declared client application service / source as 'listening' declarative stream Resource). Pluggable domain backends 'extensions’.

Containers. Services. Endpoints. Components (Metamodel, Resources). ServiceMix (OSGi), Spring, REST, RxJava / Jersey (CDI).

Streams. Functional concepts: Metamodel / Resources. DCI, Rx / Monads.

Events. Messages. Protocols. Routing / Addressing: I/O. Alignment / templates.

Uniform model 'language' for framework APIs and application instances (domains). Reified dimensional data, schema and behavior alignments rendering / activation APIs (contexts) with domains instances.

Reified Resources (Metamodel / alignments) for framework dimensional, data, schema and behavior layers entities.

DataFlow layers: publishes lower layers / subscribes upper layers entity kinds. Responsibilities, lifecycle (scopes). Discovery: subscriptions description (address, signatures, payload behaviors).

DataFlow: Node, Endpoint, Message, Exchange, Stream, Event, Stream, Exchange, Message, Endpoint, Node.

Messaging layers functional behavior: Dataflow layers uniform declarative IO / behavior (reactive / streams activation between layers) specs encoded as Resources / Metamodels: declarative dataflow graphs (layers contexts). Templates (grammars / transforms / monadic wrappers: resource 'application' as rels, operators, embeddings) for system / domains metamodels. Aggregation, alignments (metamodel / domains parser: inferred / aggregated domain assets).

Dimensional data, schema, behavior DCI. Metaclass, class, instances occurrences / contexts (templates).

Data, Information, Knowledge: dimensions / layers. Values (prices), relations (price variations), inference (price tendency).

Messaging: topics (dataflow bus) message broker. Routes (shapes / signatures: Endpoints, placeholders). Interactions.

Layers / Nodes: Dataflow Peer / Node layers (stateless bus nodes / layers specs). Declarative layer query / transforms. Contexts.

REST: Ontology backends / IO. Metamodel instances. Data.

Augmentation: Aggregation. Alignment. Activation.

OWL sameAs. Materialize alignments / schema / domains inference (OWL / RDFS ontology statements: domain / range, restrictions, role / contexts / promotion, etc.). Jena Node. Upper ontology (alignments) Nodes.

Materialize activation in actual schemas (backend connectors). Augment actual schemas / backends (backend extensions).

Temporal DB: dimensional / time travel, event sourcing, CEP. Functional immutable state: CRUD versions of records.

Hipermedia embedded semantics. Templates, rendering. Microformats. Browser (JS, REST, NodeJS) 'runat' semantics.

## Features

Specific 'helper’ nodes:

Service / backend IO.

ML / Big Data.

Integration.

IO: Nodes in Peers domains (virtualized) handling IO by activation (aggregation, alignments).

Networks:

Quad encoding. Features.

Index, Naming, Registry.

Segmentation (Messages).

Online learning (streams).

Embeddings.

Object graph (DOM) tensor encoding.

Classify: ID Merge, Aggregation.

Clustering: Attribute / rels, Forms.

Regression: Context roles, Flows.

Predictions, Purposes, Goals (Aggregation, Form, Flow) assets.

Jena: SPARQL, OWL, RDF(S) Peer.

Infer (aggregate) Peer domains into 'dynamic’ upper (common) ontology Peer.

Semiotics: Syntax (grammar / schema: sign), Semantics (sign / object, data: object), Pragmatics (behavior: concept). Logic (inference / activation, networks / statements: embeddings / operations on terms encodings). Data, schema, behavior embeddings from  
semiotic encoding features: (Context, Sign, Concept, Object).

Semantic Forms application: IDE, Browser.

Assets (Metamodel / aggregated Message context : Resource Endpoint). Data : Aggregation (augment classification : all about subjects). Schema : Forms (augment  
attributes : fill on edit about subjects). Behavior : Flows (augment context roles : show role on context selection for subject in dimension). Interaction between assets (editors).

IO Peers: Big Data. BI. Linked Data. Dimensional. Data Intelligence. SW: LDP, SoLiD, SPARQL, triple stores, etc. Lucene. NodeJS (platforms DCI activation / stubs peers).

Alignment: align class of class / instance of data / schema / behavior. Activation: alignment vocabularies 'listening’ vocab events (hierarchy of behavior, schema, data from  
abstract to concrete domains backends vocabularies. Example: raise salary, update template, translation). Augmentation (classify, attributes, roles).

Aligned activation model, materialized layers reification and dimensional resources layers: predictive / learning (rules) models, by aggregation. Example: when a salary raise is  
given.

Adding a Feature (Peer) capability to the 'bus’ add this capability (by activation augmentation) to other Peers and this Peer gets augmented by other Peers capabilities (persistence, backends, reasoning, etc.).

ETL Peer: Peer Backend IO API for declarative (templates / transforms) implementation of translation from / to different backends (datasources / services) into dimensional Metamodel Resources (statements). Aggregation, alignment, augmentation (by activation via Messages with other Peers)

Feature Peer: Peer Backend IO API for feature provider peers (alignment, augmentation, learning, reasoning, etc.). Declarative event driven augmentation pipelines by feature Metamodel activation via Messages with other Peers.

ETL Peer: Render virtual DB schema from ontology. JDBC driver. OData endpoint.

ETL Peer: Render virtual service / procedure declaration / bindings. Protocols. Export 'application' domains (dimensional data / schema / behavior). REST / SOAP / Protocol Buffers / etc.

Feature Peer: Machine Learning extensions.

Feature Peer: Alignment Metamodels (ontology Peer). Ontology matching, attribute / rel augmentation, context / role resolution, etc.

Virtualization layer (Apache Metamodel, JBoss Teiid) extension / plugin / driver.

Adding a Feature (Peer) capability to the 'bus’ add this capability (by activation augmentation) to other Peers and this Peer gets augmented by other Peers capabilities (persistence, backends, reasoning, etc.).

## Virtualization

Homogeneize backends (applications, datasources, backends). Domains. Translation of domains knowledge (dimensions, schema, data) / behavior.

Domain translation. Merge, augment and context / role alignments. Possible contexts / interactions.

Aggregation.

Alignment.

Activation.

Virtualization assets: Declarative domain templates (metamodel encoded) of aggregations, alignments, activation.

Aggregations (data), forms (schema), flows (behavior) alignments (identity / classification, attributes / clustering, roles / regression).

Index: aggregations / search.

Registry: forms / browse rels.

Naming: flows / browse state.

Possible behaviors / messages (activation flows) parsed by declarative layers templates.

Assets encoded into Metamodel resources (alignment / activation) declarative templates.

Assets dashboard.

Application kinds:

Document applications: flows, forms (questions, topics, 'tickets' kinds), QA.

CRUD / flow applications.

Alignment: align class of class / instance of data / schema / behavior. Activation: alignment vocabularies 'listening’ vocab events (hierarchy of behavior, schema, data from abstract to concrete domains backends vocabularies. Example: raise salary, update template, translation). Augmentation (classify, attributes, roles).

Aligned activation model, materialized layers reification and dimensional resources layers: predictive / learning (rules) models, by aggregation. Example: when a salary raise is  
given.

Integration as application augmentation.

Dashboards: declarative domains models (use cases: assets flows for application / services integration management).

Augmented application kinds:  
● CRUD / Flows applications.  
● Document applications: templates / forms / flows.

Domains translation: augment backends interactions. Gestures (align process definitions).

JBoss Teiid. Apache Metamodel. Backends (JDBC, REST, SOAP, JNDI, JCA, JAF, etc.): IO protocols / message streams.

RDBMS example: de normalize all schema into one table. Feed Dimensional, Data layers statements (alignment to activation: CRUD schema / behaviors).

Clients: Raw REST.

Clients: Template REST (assets).

Clients: DOM, JAF DCI stubs. JCA. Object graph (protocol buffers). Type, Content, Commands metamodel (pairs in context).

Event sourcing. Dimensional series.

Adding a Feature (Peer) capability to the 'bus’ add this capability (by activation augmentation) to other Peers and this Peer gets augmented by other Peers capabilities (persistence, backends, reasoning, etc.).

ETL Peer: Peer Backend IO API for declarative (templates / transforms) implementation of translation from / to different backends (datasources / services) into dimensional Metamodel Resources (statements). Aggregation, alignment, augmentation (by activation via Messages with other Peers)

Feature Peer: Peer Backend IO API for feature provider peers (alignment, augmentation, learning, reasoning, etc.). Declarative event driven augmentation pipelines by feature Metamodel activation via Messages with other Peers.

ETL Peer: Render virtual DB schema from ontology. JDBC driver. OData endpoint.

ETL Peer: Render virtual service / procedure declaration / bindings. Protocols. Export 'application' domains (dimensional data / schema / behavior). REST / SOAP / Protocol Buffers / etc.

Feature Peer: Machine Learning extensions.

Feature Peer: Alignment Metamodels (ontology Peer). Ontology matching, attribute / rel augmentation, context / role resolution, etc.

Virtualization layer (Apache Metamodel, JBoss Teiid) extension / plugin / driver.

Standard annotation / metadata format mechanism for activated content types augmenting content with its (learned) knowledge, accessible only with an (inferred) content identifier.  
  
Augmentation (activation): type / instance identity merge (discover type / identity equivalence of different occurrences of the same entity), attributes / links discovery (augment known schema about some entity kind and populate inferred fields), context roles (extract role from entity mention, entity from role occurrence, in a given context).  
  
Standard activation alignment: content / annotations (IO / commands / verbs).  
  
Content types: text (structured / unstructured), images, serialization formats, backends (schema annotations), domain flows (behavior annotations). Integration (dimensional annotations / metamodels).  
  
Content: identifiers, metamodel aggregation. CRUD. API (application / service).  
  
Annotations: (inferred) content identifiers relative retrievable / editable metadata (content index aligned repository). API (application / service).  
  
Client: Augmented content browser. Metamodel driven assisted content browser (sessions: purposes / goals metadata driven helper, keep browsing session items relations with each other / wizard like interface obtained from metadata). API (application / service).  
  
Although all these functionality is currently being tried to be solved by embedding 'microformats' into content, the actual approach here is those of a kind of 'protocol' for inferring / retrieving metadata identifiers via the following APIs (for example): Index (resolve metadata), Naming (resolve content), Registry (content / types / metadata bindings).  
  
So, a 'semantic' lookup for 'hollydays\_picture.png' in an HTTP resolvable context could shield to a retrievable 'hollydays\_picture.rdf' somewhere and then, perhaps, an RDF vocabulary for describing image regions could be used to describe the objects (faces) in the visualization of that file in a browser.  
  
Standardization will be needed for such approach of 'pointing' external metadata into a document. Addressing mechanisms exists for most markup languages (XPointer, XPath, XLink) and that will leverage existing XML DOM capabilities, plus XSL / XSLT declarative languages for ease of merge and transform of metadata via templates.

## Runtime

Declarative settings: nodes, routes, patterns / transforms functional flows (monads): from Metamodel schema / instances data (URIs, Resources, etc.). Metamodel hierarchy layers (endpoints / routes / messages: URIs / Resources) as common Blueprint / Spring components (schema monads classes context / roles, interactions instance data).

Messages: encoding / addressing. Signatures / patterns: dynamic discovery / flows. Bound functions: schema / domain DCI: declarative templates encoded in Metamodel (transforms).

Graph encoding (Metamodel quads): Context, SPO, Kinds. Sets (recursive statement, kind, SPOs).

Gate modelling: Dimension aggregates Measure into Flow.

Metamodel addressing: inputs / outputs of normalized URIs dataflow flowing through hierarchy from URIs till Metamodel till URIs (context driven): Aggregation, Alignments, Activation.

Backends handles Metamodel URIs IO via Metamodel metadata / functional contexts.

Runtime: generic graph functional / dataflow layered URI quads aggregation engine.

Runtime: layers. URIs IO sources / destinations. Declarative layers statements (input / output signatures: dataflow discovery).

Runtime: Metamodel hierarchy layers. Nested layers / pipelines: dataflow. Aggregation, alignment, activation.

Runtime: data driven layers declaration (pipelines / nesting). Layer schema (roles) / domain (instances). Metamodel hierarchy layers: relative in contexts.

Runtime: layers IO from outer layers to inner layers to  outer layers (URIs dimensional / contextual in / out dataflow). Queues.

Runtime: transforms (backends / services layers IO wrappers).

Runtime instantiation: specify feeds (sources) / sinks (destination) of protocol (layers / transforms / adapters) by signatures. Layers instantiate themselves into Runtime layers (pipelines, nesting by dataflow discovery / deep and wide aggregation).

Quad encoding (for runtime / graph representation): Sets of Contexts (CSPO Quads) in roles (Subject, SubjectKind, Predicate, PredicateKind, Object, ObjectKind, Statement). Objective: assign / retrieve identifiers for Quads and their components useful for algorithmic addressing, aggregation and inference (functional activation / embeddings). Available information / data: Quad elements describing a square / box, the lengths of the box sides, angles between box sides and the (four) triangles described inside the box. Obtain (spatial / dimensional) relations by geometry: inclusion, aggregation, occurrences, fitness, shapes, specializations, generalizations. Calculate Quad representative identifiers for elements (in context).

Primitives: Dimensional data, schema, behavior. Operators / relations / roles: inside / outside, properties, temporal flows / events / rels, request / response (nested contexts / IO slots, command / verbs template / context). Comparisons: octal encoding + context (rel side bit + octal digit bits). Encoding: alignment, aggregation, augmentation (activation, reasoning, inferences).

Ontology matching: data, schema, behavior (roles) represents set, subset, superset. Primitive rels / operators (transforms). Classification.

Attributes / links: clustering.

Roles / context: regression.

Exchange: addressable interactions. Event sourcing.

Make processing of RDF deterministic:

Declarative parsing (XML DOM / XSLT like): statements dataflow dimensional orders / merge / linking / alignments (signature placeholders). Canonical normalization: data model / reference model (ISO TopicMaps TMDM, TMRM, ISO-15926 like for dimensional Metamodels). Metamodel graphs encoding / representations. Correlate quads (Context, Occurrence, Attribute, Value): Occurrence as Attribute Value. Example: (Table, PK, Col, Val), semiotic (Context, Sign, Concept, Object). Add metadata for Data Model / XML DOM like Metamodel representations (DM / RM).

<http://dbooth.org/2013/well-behaved-rdf/Booth-well-behaved-rdf.pdf>

<https://json-ld.github.io/normalization/spec/index.html>

<https://www.w3.org/2018/03/jsonld-wg-charter.html>