**Contents: Mision / Vision**

Distributed Knowledge Base. Functional Syndicated Application Integration Framework. Plug existing backends (applications / datasources / services) via Connector(s) in an EAI / ESB fashion. Provide semantic augmentation of learned applications metadata (data / schema / behavior).

Problem description:

Distributed systems / micro services access to shared data. Shared data consistency. Ontology matching. Integration (EAI / ESB). Introduction of new features / products integrating over existing (linked) data with Semantic capabilities and enhancements.

Use Cases (problem / solution):

Hypermedia Use Cases (Ontology Levels). Integration / Augmentation / Alignment / Annotation of distributed resources. (Augmented) Content type driven. Encoding / Addressing (links / browse / parts / rels / roles). Microformats (embedding). Wiki like abstract representation (indexes).

Solutions:

Integration by Augmentation.

Integration by Extension.

Declarative Application Design.

Domain Business Modelling. Integration. Syndication. General purpose business domains upper ontologies for ad-hoc application building overs existing domains.

**Solution Approach**

Objectives:

Develop Protocol (APIs) to facilitate Enterprise Application Integration (EAI) by means of Semantic technologies and Machine Learning. Ontology matching driven data, schema, behavior inference / aggregation / matching. Reasoning and learning over different consolidated backends alignments for applications interoperation.  
  
Distributed P2P (Blockchain) approach of data synchronization between peers for ease of deployment patterns election and datasources integration (client APIs, microservices, etc.).

Integration by Augmention: sources / back ends. Model I/O materialized in source (plugged) application / services backends. Framework inferences augment original (source) applications and serviced.

Integration by Extension: Extended functionalities data / schema / behavior exposed as services external to source (plugged) applications. Sync (Augment / link) sources. Declaratively stated via Model descriptions. Discoverable, browseable (HAL / REST).

Declarative Application Design.

Features / Approach:

Data / Schema / Behavior Abstraction:

Source inputs of Connector(s) (plugged backends, applications, datasources) and data comming from declaratively stated Model interactions (Message IO) is rendered in a layered Model of Statement(s), each one representing: Input, Data (instance: Statement, class: Entity), Schema (instance: Kind / Role, class: Class) and Behavior (instance: Flow, class: Behavior) layers.

Layers are implemented as an RDF Quads hierarchy aggregating each one on top of another. The idea is that aggreagating Data according some criteria one could enable us to infer the Schema that those Data belongs to and that aggregating Schema and Data one could enable us to infer the Behavior (operations) that correspond to the Data manipulation in that corresponding Behavior layer class / instance.

Several types of Model(s) exists: Facets, each one preserving this layered structure. Model Facets have corresponding Layers and those layers are populated by corresponding Data, Schema, Behavior conforming Ontology Levels for each Facet. Facets abstract Model(s) inputs regarding this aspects: Source (Functional) Data, Semiotic and Dimensional Model Facets.

Facets are also populated in what are called Ontology Levels, which are Facet data, schema, behavior statements aggregated from feedback from the data, schema and behavior corresponding instance layers of the Facet Models themselves again into the input layer thus allowing for further describe upper ontology abstractions. These upper abstraction may be grouped into: Backend / Source (Data : plain inputs), Grammar / Session / Context (Schema : schema layer feedback inputs) and Interaction (Behavior : behavior layer feedback inputs).

Ontology matching (Data, Schema, Behavior alignments):

Data alignment:  
  
Determine if two instances (example: database records) of two different backends or services refer to the same entity / database row (Customers : John D. / Employees : John Doe).  
  
Schema alignment:  
  
Determine, for example, meaning and equivalences between diverse (aggregated / composite) schemas (equivalent classes / tables, equivalent attributes / columns, equivalent roles / relations).  
  
Behavior alignment:  
  
Determine meaning and equivalences between (aggregated / composite) behavior contexts and behavior contexts invocations / interactions (Appointment / Interview, anAppointment / anInterview. Behavior flows aggregated from backends / services learning).

Augmentation:

Augmentations: aggregate / align / activate (classify) sources of ontology matched data / schema / behavior enabling semantic layers interoperation.

Aggregation: Infer input data streams data, schema, behavior class / instance context layers.

Alignment: Infer layer missing / deducible attributes and values.

Activation: Infer layer CSPO Kind / Roles. Basic type system.

The idea is to infer Schema (classes and instance of classes / relations) operating over Data layer. Then, by aggregating Data and Schema, infer Behavior (classes and instances of operations / functions). This Aggregation, together with Alignment and type Activation mechanisms comprehend the Model core Augmentations.

Protocol (deployment):

Functional "Dialog" Augmentation Semantics Protocol (Dataflow Message).

Applications:

Hypermedia Dataflow Activation (reactive / event driven knowledge based contents). Dataflow layers.

Distributed: Consistency. Inference of distributed state. Event sourcing. Trust. Reconciliation.

Connected application sources (backends: EAI / ESB) and declaratively stated application models.

**Use Cases**

Domains: Use case. Problem "spaces" / domain translation / exchanges / integration.

Semantic components:

BI / EAI smart dashboards / reports / workflows / process / activity / indicators inference / prediction / execution. Abstract upper ontology application models. QA, polls, learning, profiles, guided task wizards / editors. Goal. Purpose. Forms. Templates. Model context to fulfill (roles / rels).

**RDF Introduction: Graphs, Triples, Quads**

ToDo.

**RDF for Object Graph Representations**

As RDF Quads encodes four URI values (CSPO Statement) an Object - RDF Quad elemental mapping could be implemented regarding an RDF Quad Statement CSPO as follows:

(C: Context, S: Occurrence, P: Attribute, O: Value);

where Context (C) is the URI of an Object Class identifier, Occurrence (S) is the URI of an Object Class Instance identifier and, aggregating same Class / Instance pairs, Attribute (P) and Value (O) are, respectively, Class Instance member types and values for the aggregated (S) Object of Class (C).

Contexts. Occurrences, Attributes, Values: Roles of Meta Resource(s) in contexts.

Subject in Statement has Predicate and Object Attribute / Value (roles).

Predicate in Statement has Subject and Object Attribute / Value (roles).

Object in Statement has Subject and Predicate Attribute / Value (roles).

Value as Occurrence of Attribute in Attribute Occurrence Context.

Context Kind (signature): Subject Kind and Object Kind Attribute / Value (roles).

Subject / Occurrence / Context / Role : Attribute, Value. Concepts. Semiotic Metamodel. Dimensional Encoding: each type as each (pair) kind. Pairs (tags / facets).

Meta Model: Layers Resource relations:

Instance, class, metaclass, occurrence, role. DOM, Actor / Context / Role.

Layer Context: Statement class. Aggregates same Context Statement(s). Next layer metaclass.

Layer Occurrence: Statement Context metaclass. Aggregates same Context / Occurrence Statement(s). Previous layer instance.

Layer Attribute: Statement Context Ocurrence Attribute (occurrence). Previous layer Occurrence.

Layer Value: Statement Context Occurrence Attribute Value (role). Previous layer Attribute.

Layer Aggregation begins with Model initial Statement having a new Context (class) “pushing” previous CSPO right, being the new class the new layer Context and CSP becoming SPO:

(C, S, P, O) : (N, C, S, P).

Functional / Object Oriented Resource API (Model, Statement, Semiotic, Dimensional layers, Meta Resources).

ToDo.

**Models**

Models aggregates input I/O / Connectors data into corresponding knowledge Facets (Functional, Semiotic, Dimensional).

Base Model structure / Context layers hierarchies is as follow:

Resource : Functional URI wrapper.

(Context : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);

(Statement, Occurrence, Attribute, Value);

(Entity, Statement, Occurrence, Attribute);  
(Role, Entity, Statement, Occurrence);  
(Class, Role, Entity, Statement);  
(Flow, Class, Role, Entity);  
(Behavior, Flow, Class, Role);

Models have layers in class / instance roles (except for input layer) and each upper layer aggregates functionally over the previous:

Model (Facet) Statement declaring /aggregating Model in Meta Model is of the shape:

(Model : Model Impl., Behavior, Flow, Class);

Classifying (aggregating) previous layers statements as parts of the Model.

Input Layer (Resource).

Data Layer (Statement instance, Entity class).

Schema Layer (Kind / Role instance, Class class).

Behavior Layer (Flow instance, Behavior class).

The idea is to infer Schema (classes and instance of classes / relations) operating over Data layer. Then, by aggregating Data and Schema, infer Behavior (classes and instances of operations / functions). This Aggregation, together with Alignment and type Activation mechanisms comprehend the Model core Augmentations.

**Model Layers**

What my attempts where about in the beginning was to match different URIs or, for example, database identifiers which refer to the same entity (in different databases / ontologies, for example) to perform some kind of "ontology matching".  
  
Then I've tried to develop a mechanism for using RDF Quads for encoding an object graph (and a layers class hierarchy) using Contexts to denote the class of an instance, Subjects to denote class instances and attributes (members) and values: Predicates / Objects.  
  
Quads are "reified" as Resource(s). Also, Resource is a functional wrapper reactive and event driven of an URI. And an URI could be implemented with whatever backend which could produce or consume events (databases, services, etc.). Resource layers hierarchy (Context) is to be implemented by an actor / role type object pattern according the hierarchy layer level it corresponds (and declaratively stated in a Model of Meta Resources).  
  
Then I've realized that some basic type inference could be performed with, for example, aggregating Subjects with the same predicates (Subject Kinds). Idem for Predicates, Objects and Contexts. I've also realized that plain "facts" statements could be aggregated in the previously mentioned class hierarchy to abstract further, from plain data, instance / class layers of what I call data / schema / behavior layers. Higher layers (i.e.: Behavior) "aggregate" lower layers.  
  
Layers shape is as follow:  
Resource : Functional URI wrapper.

(Context : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);

CSPO Names are according roles (Meta Resource) in the Model. For example: layer Occurrence is parent layer class.  
  
Each layer abstract instances of its own contexts instances.

Input Layer: (CSPO layer):

(Transaction, someOne, buys, someProduct);

Statement (data layer instance):

Inputs regarding the same context are aggregated into data layer instance.

(Statement, Occurrence, Attribute, Value);  
(transactionStatement, someOne, buys someProduct);

Entity (data layer class):

Aggregated Statement and Occurrence Statement occurrences reified into an Entity along with its Occurrences Attributes.

(Entity, Statement, Occurrence, Attribute);

(someTransaction, transactionStatement, someOne, buys);

Role / Kind (schema layer instance):

Aggregated Entity and Statement Entity occurrences reified into a Role / Kind along with its Statements and Occurrences.

(Role / Kind, Entity, Statement, Occurrence);  
(someBuyer, someTransaction, transactionStatement, someOne);  
  
Class (schema layer class):

Aggregated Role and Entity Role occurrences reified into a Class along with its Entities and Statements.

(Class, Role, Entity, Statement);  
(Person, someBuyer, someTransaction, transactionStatement);  
  
Flow (behavior layer instance):

Aggregated Class and Role Class occurrences reified into a Flow along with its Roles and Entities.

(Flow, Class, Role, Entity);  
(someBuy, Person, someBuyer, someTransaction);  
  
Behavior (behavior layer class):

Aggregated Class and Role Class occurrences reified into a Behavior along with its Classes and Roles.

(Behavior, Flow, Class, Role);  
(Buy, someBuy, Person, someBuyer);

Then, each Model aggregates its Statements in the form (for example):

(Model Impl, Buy, someBuy, Person);  
  
This "aggregations" are part of what I call "Augmentation(s)": Aggregation, Alignment and Activation are ones of those, which are functional transforms described declaratively in an object graph Meta Model. The act of applying an Augmentation implies one source Message Resource (context layer), one matching Template Resource (input signature) an Augmentation (Interaction functor) a Transform Resource (output signature) and a resulting (set of) Message Resource(s) materialized as further layers instances / Messages to be “parsed” by further corresponding Augmentations of matching Template signatures (dataflow).

**Model Facets**

Models have “Facets” which renders the different ways Model data / schema / behavior could be regarded and used for different purposes, from application development to Business Intelligence and Ontology Matching.

Facets are models implemented the same way other models are with Model Resource Contexts and layers and from the same data. Each Facet implements its own Resource URI wrapper (same URIs, ontology matching, provenance of aligned URIs, Facet pivoting). Then, each Facet has its own Model Context Resource class hierarchy having Augmentation / Dataflow functors as Model Resource(s) does.

Functional (Model) Facet:

Resource : Functional URI wrapper.

(Context : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);

(Statement, Occurrence, Attribute, Value);

(Entity, Statement, Occurrence, Attribute);  
(Role, Entity, Statement, Occurrence);  
(Class, Role, Entity, Statement);  
(Flow, Class, Role, Entity);  
(Behavior, Flow, Class, Role);

(Model, Behavior, Flow, Class);

Semantic / Semiotic Facet:

Resource : Functional URI wrapper.

(Context : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);

(Attributes, Occurrence, Attribute, Value);

(Object, Attributes, Occurrence, Attribute);  
(Concept, Object, Attributes, Occurrence);  
(Sign, Concept, Object, Aytributes);  
(Context, Sign, Concept, Object);  
(Interaction, Context, Sign, Concept);

(Model, Interaction, Context, Sign);

Dimensional Facet:

Resource : Functional URI wrapper.

(Context : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);

(Properties, Occurrence, Attribute, Value); Data (Properties: distance / facts).

(Value, Properties, Occurrence, Attribute); Info (Properties distance between Occurrence / previous and Occurrence / next).  
(Measure, Value, Properies, Occurrence); Knowledge.  
(Unit, Measure, Value, Properties);  
(Dimension, Unit, Measure, Value);  
(Concept, Dimension, Unit, Measure);

(Model, Concept, Dimension, Unit);

**Model Ontology Levels**

Models have “Ontology” Levels. Levels are Layers (of the Model) which are feed into its input Layer with (instance) Statements aggregated from initial input data (Data Level) aggregated into subsequent layers. Schema Level instances feeds the Model input conforming a Session (context / grammars) ontology Level. Then, behavior Level instances feeds the Model input conforming an Interaction (behavior) ontology Level.

Examples: Source, Session, Interaction declarative application protocol use case upper ontology levels (Action… Gesture, etc).

ToDo.

**URIs, Resource, Contexts Functional APIs**

Services:

Registry.

Naming.

Index.

Connectors (URIs):

JDBC.

ToDo.

**Meta Resources**

Model Context / Layers, Facets, Ontology levels, Meta Resources / Models mappings / reification. APIs. Levels example: Behavior / Interaction (Action, Gesture..., Flow). Upper ontologies: Action, Gesture etc. classes.

Contexts / Layers / Levels / Facets Meta Resources / Models classes / instances hiers (ontology matching / data, schema, behavior alignments). Members: URIs, Resource, Context, CSPO, Meta Resource / Model APIs.

Meta Resources are used by a Model Meta Model for describing models. Some of them are:

URI

Resource

Context / Context

Subject / Occurrence

Predicate / Attribute

Object / Value

Statement

Model

Kind

ContextKind

SubjectKind

PredicateKind

ObjectKind

Message

Template

Augmentation

Transform

Class

Metaclass

Instance

**Meta Models**

Meta Model: encode Layers, Contexts, Kind / Roles hierarchies (subject, context, occurrence, roles, atributes, values / metaclass, class, instance relations / meta resources).

Augmentation: Described in Meta Model. Encode Message, Template, Augmentation and Transforms roles (Meta Resources). Data, Session, Interaction Levels (Message, Template, Transform, Augmentation statements declaration realization).

Augmentation. Aggregation Meta Model: Describe layers contexts compositions. Alignment Meta Model: Describe augmented attributes (by kinds clustering). Activation Metamodel: Describe Kinds / Roles activation (by attributes aggregations).

Encoding. Addressing (contents, signatures, contexts). Events publish / subscribe. Dynamic subscriptions / bindings. Subscription, reactive Meta Resource(s). Message flow mechanism: from Model to base layers.

Match Message subjects to Templates, Augmentations and Transforms roles (bound by CK signatures dataflow). Message inputs: Models.

Reify Model Layers, Levels and Facets in a Meta Model with Meta Resources. Use Meta Resources class relations for describing models. Meta Resources describe components and roles of Models according a set of relations:

Subject (Resource) / Context (Statement) / Occurrence (CSPO instance) / Role (Kind) / Attribute / Value.

Metaclass (Occurrence) / Class (Context) / Instance (Attributes / Values).

The aim is being able to describe models using models themselves, maybe translating relations to Model Quad Statements.

The same relations could be used to build a Model in which declaratively state model dataflow behavior (reaction to events). A dataflow specification could be described by the following meta resources (roles):

Message (Subject : Data level)

Template (Context / domain : Session level)

Augmentation (Occurrence, declarative / service Resources: functors. Interaction level)

Transform (Role / range: Kind transform matches. Session level). Resulting Message Attribute / Value roles populated.

Meta Model:

Meta Resource class / instance patterns.

Participation: Subject in Occurrence.

Role: Participation for Subject.

Kind / Context hierarchies.

Subject, Participation, Occurrence, Roles, Atributes, Values / Metaclass, Class, Instance class / relations / meta resources.

(Participation, Role, Attribute, Value);

(Subject, Participation, Role, Attribute);

(Occurrence, Subject, Participation, Role);

Mappings: Facets (Models / Contexts declarations) by Meta Resource statements in Meta Model. Mappings renders Model(s) contents statements (layers) by Context Augmentations.

Augmentations defined as declarative Mappings in Meta Model encoding Context (layer) inputs matching signatures and augments current / previous layer emmiting mapping transforms. Context : Functor. Participation wraps Context / Resource.

Context::flatMap(ctx : Context) : Context

Aggregation (Augmentation): Apply each Context (layer) Functor on inputs (from input layer) and emits Transform, matching corresponding (next) layer. Next layer Context and SPO according functional mapping declared by Meta Resource types on augmented layer.

Alignment (Augmentation): ToDo.

Activation (Augmentation): ToDo.

Meta Model for Encoding / Addressing (Event routes) dataflow metadata.

ToDo.

**IDs: Addressing / Encoding**

Message - Model - Template (data) - Augmentation (functor) - Transform (interaction) - Model - Message.

Encoding, IDs: magic numbers (MIME types : Context Kinds), metaclass, class, instance, context, CSPO, etc. relations "contextual slots" for IDs. Resource resolution, Operation (primes, encoded lattice, slots context relations) factors in Meta Model relations. Encode order, hierarchies, temporal, causal (reified), containment, etc. relations into IDs encoding. Ontology matching: encoded IDs roles in context aggregation / learning.

Explain URI, Resource, Layers, Model, Kinds, etc. APIs. Meta Resources. Meta Model. Hierarchies. Order. Iteration. Flows.

Messages CRUD / Invocation semantics. Dialog. Prompts.

**Augmentation**

Reactive Context Kind (matching signatures) dataflow.

Message - Model - Template (context) - Augmentation (interaction) - Transform (data) - Model - Message.

Implementation API: Node / Container. Services (URIs Context Kind signatures resolution).

Core Services: Activation Augmentation (Naming).

Core Services: Alignment Augmentation (Index).

Core Services: Aggregation Augmentation (Registry).

Core Services: RDF / OWL Backend (endpoint, reasoning, persistence).

Core Services: DIDs Persistence (sync Node state: events sourcing).

Core Services: Protocol (I/O). Node, Session, Intetaction levels. Base Connector Augmentation API. Event driven URIs dialog / prompts protocol adapters.

Explain Context layers Aggregation Augmentation. Example: Role(s) for each CSPO. Entity in Statements. Meta Model. Meta Resources.

Explain Context layers Alignment Augmentation. Meta Model. Meta Resources.

Explain Context layers Activation Augmentation. Meta Model. Meta Resources.

Explain Augmentation. Context : Functor<Template, Transform>; Template, Transform : Context Kind (Levels: Data, Session, Interaction / Functor instance / execution contexts?).

Functors: Meta Model declarations / Context classes / instance declarative implementations. Aggregation type: invocation over each CSPO / Context roles.

Functor applied to context: Aggregation.

Functor applied to subject: Alignment.

Functor applied to predicate: Activation.

Functor applied to object: members traversal.

Message. For each layer perform each Functor: (Object : aggreg, Kind : activ, Attr : align, Obj : onto).

Augmentation:

Functors: Augmentation declaration: Meta Model definitions (Context class / instances). Message: dataflow matches Template signatures: interactions. Apply Augmentation Functors over Message contents (interactions enrich Message with Models contents: ontology matching / Levels / Facets). Materialize / emit dialog / prompts Message (enrich Message from Models / reactive IO events).

Augmentation: Context / Functors. Message Resource(s) / Meta Resource(s) (nested / wrapped) elements determines flow Template Transform results / behaviors (CRUD, Functor invocations). Message IO performs Augmentations. Ontology levels resolution (Templates / Transforms / Augmentatiom levels: matching patterns / dialog prompts in Ontology levels).

ToDo.

**Dataflow**

Explain Context layers Aggregation Augmentation. Example: Role(s) for each CSPO. Entity in Statements. Meta Model. Meta Resources.

Explain Context layers Alignment Augmentation. Meta Model. Meta Resources.

Explain Context layers Activation Augmentation. Meta Model. Meta Resources.

Explain Augmentation. Context : Functor<Template, Transform>; Template, Transform : Context Kind (Levels: Data, Session, Interaction / Functor instance / execution contexts?).

Functors: Meta Model declarations / Context classes / instance declarative implementations. Aggregation type: invocation over each CSPO / Context roles.

Functor applied to context: Aggregation.

Functor applied to subject: Alignment.

Functor applied to predicate: Activation.

Functor applied to object: members traversal.

Message. For each layer perform each Functor: (Object : aggreg, Kind : activ, Attr : align, Obj : onto).

Augmentation:

Functors: Augmentation declaration: Meta Model definitions (Context class / instances). Message: dataflow matches Template signatures: interactions. Apply Augmentation Functors over Message contents (interactions enrich Message with Models contents: ontology matching / Levels / Facets). Materialize / emit dialog / prompts Message (enrich Message from Models / reactive IO events).

Augmentation: Context / Functors. Message Resource(s) / Meta Resource(s) (nested / wrapped) elements determines flow Template Transform results / behaviors (CRUD, Functor invocations). Message IO performs Augmentations. Ontology levels resolution (Templates / Transforms / Augmentatiom levels: matching patterns / dialog prompts in Ontology levels).

Interactions declarations: signature definitions (Template / Transform contexts). Interaction instances: addressable exchanges (Augmentations, Message, Model context / Mapping bindings / matchings / performances). Meta Model / Levels event driven Model Augmentation.

Interactions declarations: signature definitions (Template / Transform contexts). Interaction instances: Exchanges (Augmentations, Message, Model context / Mapping bindings / matchings / performance). Contexts / Exchanges: Meta Model / Levels event driven source Augmentation events declarations (populating Facets / Layers / Levels).

ToDo.

**Ontology Matching**

Ontology matching. Dataflow: sort statements. Units. Equivalences. Distances / events (order). Services (Augmentation / Context Functors Meta Model mappings / transforms).

Explain ontology matching: data, schema, behavior alignments. Layers. Levels. Facets. Meta Resources / Model. IDs, Encoding / Addressing.

Encoding. Functional, Semiotic, Dimensional (Facets). Layers. Levels. Meta Resource / Model. Sets. Value as occurrence of attribute. metaclass / class / instance IDs.

Functionsl / Semiotic / Dimensional layers / levels examples / alignments.

Ontology matching (Data, Schema, Behavior alignments):

Data alignment:  
  
Determine if two instances (example: records) of two different backends or services refer to the same entity (Customers : John D. / Employees : John Doe).  
  
Schema alignment:  
  
Determine, for example, meaning and equivalences between diverse (aggregated / composite) schemas (equivalent classes / tables, equivalent attributes / columns, equivalent roles / relations).  
  
Behavior alignment:  
  
Determine meaning and equivalences between (aggregated / composite) behavior contexts and behavior contexts invocations / interactions (Appointment / Interview, anAppointment / anInterview. Behavior flows aggregated from backends / services learning).

**Implementation**

Persistence:

Interaction Model: aggregated Meta Model interactions (performed / inferred / possible) declared Models events (saga pattern).

Languages, Patterns, APIs, Frameworks. (Container, Node, Model, Service, etc.).

Deployment / Implementation:

Messaging / Storage:

Vert.x.

Fuse / ServiceMix.

Jena.

DIDs (ont.io).

Container / Node:

Java (Spring / OSGi).

JavaScript (NodeJS).

Protocols:

XML / XSL. Event bus (encoding / discovery). Addressing (node / model / ontology levels, topics / queues).

Services:

Registry.

Naming.

Index.

Connectors:

JDBC.

ToDo.

**Deployment**

Protocol plugins (Protocol Service) Connectors. Runtime. Core Services. Endpoints. Dataflow.

**To Do**

* Model, URIs, Resource, Contexts Functional APIs. Meta Model / Resources encoding. Mappings.
* IDs: Addressing / Encoding. Semantic (signature, contents, context) resolvable / discoverable identifiers.
* Ontology Matching:
* Ontology Matching: IDs, Addressing, Encoding. Functional, Semiotic, Dimensional (Facets). Layers. Levels. Meta Resource / Model. Sets. Value as occurrence of attribute. metaclass / class / instance IDs.
* Ontology Matching: Encode: order, iteration, flows, units, relations, events, enums, etc.
* Meta Resource / Meta Model:
* Meta Resource / Model: encode Model, URIs / Layers / Contexts / Facets / Levels / Resources hierarchies. Mappings.
* Meta Resource / Model: Encode Message, Template, Augmentation(s), Transforms and Mappings (Dataflow).
* Meta Model: Data, Session, Interaction Levels (Message, Template, Transform, Augmentation statements). Mappings.
* Messages: Events IO / Persistence: Saga Activation / Passivation populating Node local Quad store / persisting peers via DIDs ([ont.io](http://ont.io)) semantic (resolvable / discoverable) identifiers.
* Messages: Mappings. Meta Resources / Model Message based Model interactions (Subscriptions).
* Messages: Message semantics (Augmentation: Verbs, CRUD, Behavior) according Message structure / pattern (dialog / prompts).
* Messages: Dataflow Template matches signatures (Session level, enrichs Message with Model / Dialog prompts / contents). Augmentation Functor applied over Message contents (Interaction level). Transform matching output signature emits (Session level, populated / prompts) output Message.
* Messages: Dataflow. Subscriptions. Reactive Model. Dynamic subscriptions / bindings. Events publish / subscribe between Model Resource. Mappings.
* Messages: Saga Activation. Interaction Model (Meta Model). Aggregated (Interaction) Meta Model interactions (performed / inferred / possible) emitted as Model event Messages (Saga pattern). Mappings.
* Messages: Saga Passivation. Model layers data routed by Mappings as event Message into (Interaction) Meta Model. Message inputs: Models. Mappings. Populate
* Augmentation:
* Functors: Meta Model declarations / Context classes / instance declarative implementations. Aggregation type: invocation over each CSPO / Context roles.
* Functor applied to context: Aggregation.
* Functor applied to subject: Alignment.
* Functor applied to predicate: Activation.
* Functor applied to object: members traversal.
* Meta Model:
* URI;
* Resource (URI);
* Item (Type, Resource) : Resource;
* Statement (Item) : Item;
* Context (Resource, Statement) : Statement;
* (Context, Statement, Item, Resource);
* (Class, Metaclass, Instance, Occurrence);
* State class hierarchies / Model (Facets) Resource types.
* State Facets / Layers / Levels Mappings.
* Aggregations:
* Kind in Context of a Class has Entity as Predicate.
* Employee in Context of Boss has corresponding salary in Attribute / Value.
* Order (ctx class hiers axes), comparisons, iterations, flow, events, rels, units, equivalence, etc.
* Compare common Statement / Context (Alignment matching): (Context, Statement, Item, Resource);
* Functional API: Message IO. Mappings.