**Mision / Vision**

**Objectives**

**Description**

**Use Cases**

**Problems**

**Solution**

**Approach**

**Features**

**Reactive event driven**

**Augmentation Dataflow**

**Ontology Matching**

**RDF Introduction**

**RDF Objects mapping**

**Models**

**Layers**

**Interaction Layer**

**Data**

**Schema**

**Behavior**

**Meta Model**

**IDs: Encoding**

**Facets / Levels**

**Dataflow**

**Resource APIs: Monad(OntResource)**

**Model functional APIs: factories**

**Reactive streams (kinds, etc.)**

**Message: Monad(Resource)**

**Model Functional APIs : Message builder**

**Functional APIs: reactive streams (Augmentation / dynamic routes)**

**Persistence**

**Dataflow**

**Model Interaction Layer matches Messages**

**Interaction Layer Augmentation dataflow**

**Meta Resources (Template, Transform) / Transform dataflow embeddings**

**IDs: Encoding**

**Messaging: Addressing / Discovery**

**Augmentation: Functor(Message, Message) : Event**

**Dataflow Model Events declaration**

**Meta Resources: Interaction IO (Messages Meta Resources bindings / embeddings / prompts / mappings)**

**Aggregation**

**Alignment**

**Activation**

**Service**

**Connector**

**Ontology Matching**

**Index**

**Registry**

**Naming**

**IDs: Encoding**

**Ontology Matching**

**Augmentation Services**

**ToDo**

**Mision / Vision**

**Objectives**

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

ToDo.

**Description**

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).

ToDo.

**Use Cases**

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.

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

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).

Hypermedia Activation. Addressing. Link extended content types resources elements / parts with other resources addressed elements.

Domains: Use cases domains abstractions (problem "spaces" / ontologies) enabling 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).

ToDo.

**Problem**

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).

ToDo.

**Solution**

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.

ToDo.

Description:

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.

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):

**Approach**

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 DIDs) 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 services.

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**

### Ontology matching

Determine whether two identifiers refer to the same entity, whether two relations are the same and which results corresponds to instances of the same actions.

ToDo.

### Augmentation Protocol

Functional declarative way of stating Augmentation Transforms over Messages / Resources matching / populated by input Templates performing output Mappings Augmentation reflecting input, model and behavior state.

ToDo.

### Reactive / Event Driven

Message based Augmentation Events Dataflow. Augmentation Mapping Dataflow allowing to embed dynamic state in Model entities (including Mappings Augmentations themselves).

ToDo.

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).

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.

ToDo.

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).

Trust. Consistency. Event sourcing. Inferencing (of distributed state). Reconciliation.

Certify Entity / Subject Identity. Class / instance alignment (matching).

Integration: Augment sources / back ends. Model I/O materialized in source (plugged) application / services back ends.

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

Augmentations:  
  
Activation (type inference): classification (determine class / metaclass / roles for entity attributes and values).  
  
Alignment (infer attributes / relations): clustering (from multiple occurrences of same entity in diverse data sources).  
  
Aggregation: infer roles in contexts: regression (Person class in Employment interaction : Developer role).  
  
Integration of addressable resources. Reactive I/O (sync back ends). Content type driven semantic augmentation / annotations.

Augmentation of distributed resources. Annotations (Semantic / ML). API for resource / schema / interactions exploration / protocol for message based API "dialogs" execution. HAL (Hypertext Application Language), OData (REST) like interfaces.

Hypermedia Activation. Addressing. Link extended content types resources elements / parts with other resources addressed elements.

Domains: data, schema and behavior of business applications (ERP, CRM, BI, SCM, HMS, etc.).

General purpose business domains problem resolution / tasks, goals accomplishment helper tools.

Syndication (contextual hypermedia activation): QA. Polls. Learning. Profiles. Guided task (wizards), guided editors: Context: Goal / Purpose.

# RDF Introduction: Graphs, Triples, Quads

ToDo.

# RDF Quads 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 (name, domain / range) 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 (Occurrence)..

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

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**

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

Base Model structure / Context layers hierarchies is as follow:

OntResource (URIs).

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); Interaction / Meta Model.

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.

Models:

URI(s);

OntResource; Merged URI(s) wrapper.

Resource (OntResource CSPO / Contexts hierarchies Monad wrapper);

Message (Resource Monad wrapper); Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

Augmentation : Functor.

Mappings: Declarative IO signatures: Context Kinds Templates / Transforms. Subscriptions / routes. Dataflow.

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors behavior encoding statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

Interaction Model:

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

**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.

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.

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).  
  
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); Interaction / Meta Model.  
  
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.

Meta Model:

URI;

Resource (URI\*);

Role (Model CSPO hierarchies) : Resource;

Statement (Resource, Resource, Resource, Resource) : Resource;

Kind (Statement\*) : Resource;

Class (Kind\*) : Resource;

Context (Class\*) : Resource;

Hierarchy: class (Object / Value) as superclass Context.

Object: class (extension);

Context: super class (intention);

(Kind, Statement, Role, Resource); Data (Resource Kind).

(Class, Kind, Statement, Role); Schema (Role Class)

(Context, Class, Kind, Statement); Interaction (Statement Context).

State Facet / Layer / Level / Augmentation / Model Resource Mappings.

Functional API: Message IO. Mappings.

State order (in context class hierarchies axes), comparison relations, iterations, flow, events, causal relations, units, enums, equivalence, etc.

Data order: Resource Kind hierarchies.

Schema order: Role Class hierarchies.

Interaction order: Statement Context hierarchies.

ToDo.

**Meta Resources**

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.

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 Model:

URI;

Resource (URI\*);

Role (Model CSPO hierarchies) : Resource;

Statement (Resource, Resource, Resource, Resource) : Resource;

Kind (Statement\*) : Resource;

Class (Kind\*) : Resource;

Context (Class\*) : Resource;

Hierarchy: class (Object / Value) as superclass Context.

Object: class (extension);

Context: super class (intention);

(Kind, Statement, Role, Resource); Data (Resource Kind).

(Class, Kind, Statement, Role); Schema (Role Class)

(Context, Class, Kind, Statement); Interaction (Statement Context).

State Facet / Layer / Level / Augmentation / Model Resource Mappings.

Functional API: Message IO. Mappings.

State order (in context class hierarchies axes), comparison relations, iterations, flow, events, causal relations, units, enums, equivalence, etc.

Data order: Resource Kind hierarchies.

Schema order: Role Class hierarchies.

Interaction order: Statement Context hierarchies.

**Meta Model**

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.

Meta Model:

URI;

Resource (URI\*);

Role (Model CSPO hierarchies) : Resource;

Statement (Resource, Resource, Resource, Resource) : Resource;

Kind (Statement\*) : Resource;

Class (Kind\*) : Resource;

Context (Class\*) : Resource;

Hierarchy: class (Object / Value) as superclass Context.

Object: class (extension);

Context: super class (intention);

(Kind, Statement, Role, Resource); Data (Resource Kind).

(Class, Kind, Statement, Role); Schema (Role Class)

(Context, Class, Kind, Statement); Interaction (Statement Context).

State Facet / Layer / Level / Augmentation / Model Resource Mappings.

Functional API: Message IO. Mappings.

State order (in context class hierarchies axes), comparison relations, iterations, flow, events, causal relations, units, enums, equivalence, etc.

Data order: Resource Kind hierarchies.

Schema order: Role Class hierarchies.

Interaction order: Statement Context hierarchies.

Models:

URI(s);

OntResource; Merged URI(s) wrapper.

Resource (OntResource CSPO / Contexts hierarchies Monad wrapper);

Message (Resource Monad wrapper); Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

Augmentation : Functor.

Mappings: Declarative IO signatures: Context Kinds Templates / Transforms. Subscriptions / routes. Dataflow.

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors behavior encoding statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

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.

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.

Messages CRUD / Invocation semantics. Dialog. Prompts.

Encoding: Cons lists. Trees. Huffman / Prefix codes. RDF List serialization. Meta Resources / Models declarative Encoding, Addressing, Mappings, Transforms (Immutable sequences, dataflow Mapping: Template / Augmentation / Transform functional streams).

Kinds, Signatures. Contents. Contextual metadata. Sets (bitstring cuads). Lattices.

(C (S (P (O, Nil))));

(C2 (C (S (P, Nil)));

Models:

URI(s);

OntResource; Merged URI(s) wrapper.

Resource (OntResource CSPO / Contexts hierarchies Monad wrapper);

Message (Resource Monad wrapper); Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

Augmentation : Functor.

Mappings: Declarative IO signatures: Context Kinds Templates / Transforms. Subscriptions / routes. Dataflow.

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors behavior encoding statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

**Messages: Model 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 / Mappings).

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.

Models:

URI(s);

OntResource; Merged URI(s) wrapper.

Resource (OntResource CSPO / Contexts hierarchies Monad wrapper);

Message (Resource Monad wrapper); Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

Augmentation : Functor.

Mappings: Declarative IO signatures: Context Kinds Templates / Transforms. Subscriptions / routes. Dataflow.

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors behavior encoding statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

ToDo.

Models aggregates Message input IO / Connectors data into corresponding knowledge Facets (Functional, Semiotic, Dimensional). Model is a layered structure of RDF Quads which follow the base (Functional) Model structure:

OntResource is the class responsible for aggregating different URIs referring the same entities (Ontology Matching).

Resource : Functional (Monad) OntResource 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 layer statements in which statement context (Facet Roles) classes are a hierarchy from Resource to Behavior and where context role instances follow a hierarchy of a dynamic type system (Kinds).

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.

ToDo.

## Model Abstraction

### Models: 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.

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.

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).  
  
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); Interaction / Meta Model.  
  
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).

ToDo.

### Models: 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:

(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 Model Facet:

(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 Model Facet:

(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);

ToDo.

### Models: 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.

#### Model Source Level (Backend)

Input Statements coming from plain RDF Quads aggregated according Data / Schema / Layers Augmentation(s). Base facts Model Level.

ToDo.

#### Model Session Level

Aggregate Source (Backend) Level Schema layer Statements as Model Session level Data layer input. Reify Schema (roles / grammars).

ToDo.

#### Model Interaction Level

Aggregate Session Level Behavior layer Statements as Model Data level Data layer input. Reify behaviors (context / interactions).

Declarative application protocol use case upper ontology levels (Action… Gesture, etc).

ToDo.

## Meta Resources

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

(Augmentation, Template, Mapping, Transform); Meta Resources.

Model declared as Interaction Model Augmentation (matching Messages) in Interaction Model. Flows:

Message

Model (Functor)

Augmentation (Addressable Interaction)

Template (Message)

Mapping (Functor)

Transform (Message)

Model (Functor)

Message

Interaction Model: Model Events (Augmentation).

Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output.

Embeddings: Message Match Event Signature. Tempate matching / Transform rendering. Dataflow.

Embeddings: Meta Model Augmentation, Template, Mapping, Transform Meta Resources (input layer). Meta Model Source, Session, Interaction levels.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

ToDo.

## Meta Model

Meta Model: encode Layers, Contexts, Kind / Roles hierarchies (subject, context, occurrence, roles, atributes, values / metaclass, class, instance relations / meta resources) and Facets using corresponding Facets implementations of base Model Meta Resources.

URI(s);

OntResource; Merged URI(s) wrapper.

OntResource hierarchy: layers statement contexts. Facets DOM, Actor / Role.

Resource (OntResource Context Roles hierarchies Monad wrapper);

Resource (OntResource CSPO / Contexts hierarchies Monad wrapper);

Role (Model CSPO Context Roles hierarchies type classes) : Resource;

Statement (Resource, Resource, Resource, Resource) : Resource;

Kind (Statement\*) : Resource;

Class (Kind\*) : Resource;

Context (Class\*) : Resource;

Hierarchy: class (Object / Value) as superclass Context.

Object: class (extension);

ContextStatement: super class (intention); Context Role.

(Resource, ?, ?, ?);

(Role, Resource, ?, ?);

(Statement, Role, Resource, ?);

(Kind, Statement, Role, Resource); Data (Resource Kind).

(Class, Kind, Statement, Role); Schema (Role Class)

(ContextStatement, Class, Kind, Statement); Interaction (Statement ContextStatement).

State Facet / Layer / Level / Augmentation / Model Resource Mappings.

Functional API: Message IO. Mappings.

ToDo.

## Interaction Model

(Augmentation, Template, Mapping, Transform);

Model declared as Interaction Model Augmentation (matching Messages) in Interaction Model. Flows:

Message

Model (Functor)

Augmentation (Addressable Interaction)

Template (Message)

Mapping (Functor)

Transform (Message)

Model (Functor)

Message

Interaction Model: Model Events (Augmentation).

Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output.

Embeddings: Message Match Event Signature. Tempate matching / Transform rendering. Dataflow.

Embeddings: Meta Model Augmentation, Template, Mapping, Transform Meta Resources (input layer). Meta Model Source, Session, Interaction levels.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

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.

Interaction Model for Encoding / Addressing (Mapping : Event routes) Dataflow metadata.

Augmentation: Described in Interaction 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).

ToDo.

## Dataflow: Mapping

Interaction Model declares Events (Augmentations) which have a functional Mapping between its domain (Template) and range (Transform). An Augmentation Context Kind correspond to this Mapping “signature”. Dataflow binds input Message(s) to domain Template by pattern matching and resolving any input Message references (Addressing).

Outputs are resolved by pattern matching with Transform, Message and existing Model data. Augmentations may play the role of “placeholder” Resource(s) which are bound to context aware Augmentations thus rendering Transforms into Model entities (including Mapping Augmentations themselves).

Model declared as Interaction Model Augmentation (matching Mappings) in Interaction Model. Flows:

Message

Model (Functor)

Augmentation (Addressable Interaction)

Template (Message)

Mapping (Functor)

Transform (Message)

Model (Functor)

Message

Interaction Model: Model Events (Augmentation).

Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output.

Embeddings: Message Match Event Signature. Tempate matching / Transform rendering. Dataflow.

Embeddings: Meta Model Augmentation, Template, Mapping, Transform Meta Resources (input layer). Meta Model Source, Session, Interaction levels.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

ToDo.

**Resource API**

Model Functional APIs. Reactive streams. Meta Resource factory (Template / Transform dataflow placeholders).

Resource<T> : T extends OntResource (Context Roles hierarchy, normalized aligned URIs wrapper).

Functor<Resource<T>, Resource<T>>;

Message<Resource<T>>;

Resource<T> : T extends OntResource (Context Roles hierarchy, normalized aligned URIs wrapper).

Functor<Resource<T>, Resource<T>>;

Message<Resource<T>>;

**RDF triples, quads introduction**

RDF Models: rdfs type, class, subClassOf, sameAs, reification when appropriate. RDFS. OWL (alignments).

RDF / OWL Backend: APIs. Details: Contents triples / models introductions.

Turtle. N3.

Example: feed Dimensional model for equivalences (units), comparison (orders).

TBD.

**Models: Quads, Contexts, Occurrences, Attributes, Values.**

Declarative means of using RDF quads to state application object models (data, schema and behavior).

Aggregation.  
Kinds.  
Grammar.

Formalization: Functional / Object API. Reference / Data model. Sets, categories, models.

Subjects: attributes / values, contexts / roles.

(Context, Occurrence, Attribute, Value);  
(Context, Sign, Concept, Object);

Instance, occurrence, class, metaclass.

Hierarchies: layered quad statements are represented by a class hierarchy which root is the Resource<T> monad. There is a subclass relationship between each layer implementing class and the one of the next layer (Dynamic Object Model).

Quads in the context role of lower layers represents occurrences of context enclosing layer. Assert class hierarchies, order relation (temporal, causal, containment, etc.) by attrs / vals, set / superset relations.  
Discovery: All model kinds are browseable / discoverable.

Determine class (reified layers contexts) hierarchies:  
(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model resources, statements, kinds. Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch,  
event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.  
Dimensional / Grammar models.

TBD.

**URIs, Resource, Statement, Kind APIs**

TBD.

Message service URIs: contextual (statement / dialog) service invocations.

Example: Subject (image URI / resource : source), Predicate (detection service / index service), Object (detection / search results endpoint / placeholder : destination).

Grammars: Predicate Kind (face / search recognition signature) from Subject (faces images / names) / Object (face classes / subjects) Kinds. Kind model layers.

Models definition: data (Statement, Entity), schema / context (Role, Class), interactions / behavior (Flow, Behavior).

Kinds / Roles:  
Grammar: kinds layers aggregation (CSPO layers Kinds).  
Layers: Roles (Models metaclass context resources).

Reified Kind: (Kind, Occurrence, Attribute, Value);  
Grammar input set model specificatíon (Statement layer kinds).

Dimensional input set model specificatíon (from Statement layer, ordered SPOs: order criteria, comparisons. Kinds / class / occurrence / instance  
order criteria?). Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements).

MetaGraph (resolution). Dimensional / Grammar alignments / annotations.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch,  
event bus routes. URIs / IDs mappings. Resource set specification resolution. MetaGraph resolves concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics via MetaGraph driven transforms (data / schema / behavior augmentation: dialogs).

URIs API for annotating network retrievable resources metadata. Content type / model driven augmentations / activations (models features / outputs). Subject attributes / values. Occurrences contexts / roles. Paths, pointers, locators. Example: annotate document URIs (parts, sections, mentions), annotate images URI (whole image description, coords: classes, individuals), annotate DB, table, row, column, value URIs, annotate / describe service / APIs URIs. Hypermedia protocol composable with other (described / annotated) APIs / resources. Example: Drive APIs.

Resource<T : URI> monad. Message functors. Transform reactive extensions.

Transform : Observer / Observable of Resource<T : URI>. Stream. Built upon Resources / Messages (TransformBuilder).

Identity and other core transforms (core messages). Stream.

flatMap(Message::apply) : Transform<Resource<R : URI>>.

API: Class for layer for model.  
API: Class for layer (DOM).  
API: Parameterized Resource: layer classes determined by URIs hierarchy, i.e.: Resource<Entity>, Entity : URI.

Base core services URIs (index, naming, registry). URI subclasses implementing / wrapping state for Resource monads offering protocols / addressing / content types / representations facades for services: DBs, WS (REST, SOAP, SPARQL), ML (predictions), etc.

Discovery: All model kinds are browseable / discoverable.

Determine class (reified layers contexts) hierarchies:  
(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model resources, statements, kinds.

Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

MetaGraph (resolution). Dimensional alignments / annotations.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

**Models**

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

Base Model structure / Context layers hierarchies is as follow:

OntResource (URIs).

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 classifying and 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 API:

URI(s);

OntResource; Merged URI(s) wrapper (Ontology Matching).

Resource (OntResource CSPO / Contexts hierarchies Monadic functional wrapper);

Statement (Resource Quad) : Resource.

Kind. Context Facets hierarchy roles.

Message (Resource Monad wrapper); Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

Augmentation : Functor<Template, Transform>.

Mappings: Augmentation Context Kind declarative IO signature. Templates / Transforms. Subscriptions / routes. Dataflow.

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors behavior encoded statements. Events: Augmentation. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

Interaction Model Layer:

(Template, Mapping, Transform, Model);

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

**Meta Resources**

(Template, Mapping, Transform, Model);

(Augmentation, Template, Mapping, Transform); Meta Resources.

Model declared as Interaction Layer Augmentation(s) (matching Messages) in Interaction Model. Flows.

Transform result adds new Context / Occurrence, merges existing Context Occurrence in Model with new Statements (prompts / embeddings). Dataflow.

Augmentation: Event (Augmentation) emits new dataflow Message Transforms from Template Mappings.

Augmentation: Input Templates prompts. Dialog, Meta Resource embedded Templates / Mappings / Augmentations in input Message / output Transform.

Embeddings: Meta Model Augmentation, Template, Mapping, Transform Meta Resources (input layer). Meta Model Source, Session, Interaction levels.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

**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.

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.

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).  
  
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); Interaction / Meta Model.  
  
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).

One also could Augment Resource(s) in a functional manner, using reactive event driven APIs so, for example applying "Person" class to "Employee" role could shield a Resource set of people being working for someone. The ultimate goal is to be able to "plug" as much "backends" connectors as posible into distributed peers which exposes protocols / APIs for knowledge driven hypermedia applications.

**Kinds**

Kinds (Application):

Kind: Basic type inference. Applied over layers CSPO during Activation Augmentation. An Occurrence Attributes / Values, aggregated for its URI and Context, determines Kind "members" (Attribute) and Kind instance member values (Value).

Super Kind / sub Kind hierarchy relationship is given by a set of Kind Attributes being super set / sub set of each other.

Examples.

SubjectKind (meta Resource): For a given URI occurring as Subject (Occurrence) across a set of Statements (Contexts), its aggregated Predicates (Attributes) defines its "Kind" and its Attribute values determines the given Kind instance "members" values.

ObjectKind (meta Resource): for a given URI occurring as Object (Value) over a set of Statements, Subject (Kind Attribute), Predicate (Kind Value).

PredicateKind (meta Resource): for a given URI occurring as Predicate over a set of Statements, Object (Kind Attribute), Subject (Kind Object).

ContextKind: SubjectKind (Attribute), ObjectKind (Value). Context (Statement) "signature" (dataflow inputs / outputs activation: domain / range).

**Functional Model**

OntResource is the class responsible for aggregating different URIs referring the same entities (Ontology Matching).

Resource : Functional (Monad) OntResource 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);

**Semiotic Model**

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

(Attributes, Occurrence, Attribute, Value);

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

(Model, Interaction, Context, Sign);

**Dimensional Model**

(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, Properties, Occurrence); Knowledge.  
(Unit, Measure, Value, Properties);  
(Dimension, Unit, Measure, Value);  
(Concept, Dimension, Unit, Measure);

(Model, Concept, Dimension, Unit);

Dimensional alignment / aggregation layers (lower resource alignment layers):

(Value, Distance, Prev, Next : in Units); (Measure, Value...) (Unit, Measure, Value,...); (Resource, Unit, Measure, Value); Marriage event example.

Model Contexts: Meta Resources / Contexts hierarchies. Models:

Data: Source / Interaction, Schema: Encoding / Grammar, Behavior: Dimensional / Measures (marriage).

Example:

(Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.

Order layers statements. Hierarchies (contexts / kinds). Parent / child relationships (steps). Order type relationships: husband: single / marriage / married.

(Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.  
(Measure, Value, Previous, Distance);  
(Unit, Measure, Value, Previous);  
(Dimension, Unit, Measure, Value);  
(Concept, Dimension, Unit, Measure);  
(Resource, Concept, Dimension, Unit);  
(Statement, Resource, Concept, Dimension);

Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements layers).

Value -> distance(prev, next); ordering;

Assert knowledge: 1h -> 60min;

dom-lun-mar-mie-jue-vie-sab (orders);

1mt -> 100cm;

etc.

Comparison / order: Alignments (prev, curr, next asserted knowledge). Next hour, location, city, country, next distance at next time at current speed. Event sourcing / tracking: married -> marriage occurred.

Sort: cause / effect, temporal, etc. Messages align, functional map, fold, etc. Primitives. Encode layered statements ordering. Complement / supplement concepts definitions.

Encoding / Dimensional example: role in context. X is Y for Z in W.

(W (Z (X (Y))));

(Marriage (Role (Man (Husband))));

(Hour (Minute (1 (60))));

Encoding, Dimensional, Meta Model. Units. Events. Order. Relations. Comparison. Input layers. Augmentation.

Events metamodel (TBD):

**Model Levels (upper ontologies)**

Source: Data / Facts

Session: Grammar / Contexts

Interaction: Gestures / Actions

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).

Metagraph / Grammar (sample):

(Kind, SuperKind, Attribute, Value);

(Occurrence, Kind, SuperKind, Attribute);

(Context, Occurrence, Kind, SuperKind); (attributes / links bindings).

(Resource, Context, Occurrence, Kind); State Resource Kind in occurrence context (context / role bindings).

(Statement, Resource, Context, Occurrence); State Resource URIs occurrences / Resource class IDs (classification bindings).

(Interaction, Statement, Resource, Context);

(Action, Interaction, Statement, Resource);

Interaction / Model?

Action / Schema?

**Interaction Layer**

Model declared as Interaction Layer Augmentation(s) (matching Messages) in Interaction Model. Flows.

Transform result adds new Context / Occurrence, merges existing Context Occurrence in Model with new Statements (prompts / embeddings). Dataflow.

Augmentation: Event emits new dataflow Message Transforms from Template Mappings.

Augmentation: Input Templates prompts. Dialog, Meta Resource embedded Templates / Mappings / Augmentations in input Message / output Transform.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

(Template, Mapping, Transform, Model);

(Augmentation, Template, Mapping, Transform);

(Model / Template): class / superclass (intension / extension).

Model declared as Interaction Model Augmentation (matching Messages) in Interaction Model. Flows (levels):

Message

(\*) Model (Augmentation Functor)

Template (Message) : populate / prompt

Mapping (Functor)

Transform (Message) : populate / prompt

(\*) Model (Augmentation Functor)

Message

(\*): Feedback.

Interaction Model: Model Events (Augmentation).

Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output.

Embeddings: Message Match Event Signature. Tempate matching / Transform rendering. Dataflow.

Embeddings: Meta Model Augmentation, Template, Mapping, Transform Meta Resources (input layer). Meta Model Source, Session, Interaction levels.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

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.

Interaction Model for Encoding / Addressing (Mapping : Event routes) Dataflow metadata.

Augmentation: Described in Interaction Layer. 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).

**Meta Model**

URI(s);

OntResource; Merged URI(s) wrapper.

OntResource hierarchy: layers statement contexts. Facets DOM, Actor / Role.

Resource (OntResource Context Roles hierarchies Monad wrapper);

Resource (OntResource CSPO / Contexts hierarchies Monad wrapper);

Role (Model CSPO Context Roles hierarchies type classes) : Resource;

Statement (Resource, Resource, Resource, Resource) : Resource;

Kind (Statement\*) : Resource;

Class (Kind\*) : Resource;

Context (Class\*) : Resource;

Hierarchy:

Context: class (intention);

Object: super class (extension);

(Resource, ?, ?, ?);

(Role, Resource, ?, ?);

(Statement, Role, Resource, ?);

(Kind, Statement, Role, Resource); Data (Resource Kind).

(Class, Kind, Statement, Role); Schema (Role Class)

(ContextStatement, Class, Kind, Statement); Interaction (Statement ContextStatement).

State Facet / Layer / Level / Augmentation / Model Resource Mappings.

Functional API: Message IO. Mappings.

**IDs : Encoding / Addressing**

Encoding of interaction layer matches Message(s) with Model (Levels) contents (patterns / hierarchies / dialogs).

IDs:

URI(s);

OntResource; Merged URI(s) wrapper.

OntResource hierarchy: layers statement contexts. Facets DOM, Actor / Role.

Resource (OntResource Context Roles hierarchies Monad wrapper);

Statement : Resource quad, Resource.

Message (Resource Monad wrapper); Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

IDs:

A: OntResource.

B: CSPO Role.

C: Statement : OntResource Occurrence.

D: Kind CSPO Instances.

E: Class : Kind CSPO Classes.

F: ContextStatement : Context Role.

Meta Model:

A: (Resource, ?, ?, ?);

B: (Role, Resource, ?, ?);

C: (Statement, Role, Resource, ?);

D: (Kind, Statement, Role, Resource); Data (Resource Kind).

E: (Class, Kind, Statement, Role); Schema (Role Class)

F:.(ContextStatement, Class, Kind, Statement); Interaction (Statement ContextStatement).

ID: (F (E (D (C (B (A, Nil))))));

Cons lists. Binary Trees. Huffman / Prefix codes. RDF List serialization. Meta Resources / Models declarative statements Encoding, Addressing.

Kinds, Signatures. Contents. Contextual metadata. Lattices. Roles.Sets (bitstring cuads). Definitions (elements). Operations. Rules. Categories. Groups.

Statement: (ID (ID (ID (ID, Nil))));

(C (S (P (O, Nil))));

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.

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.

Messages CRUD / Invocation semantics. Dialog. Prompts.

Encoding: Cons lists. Trees. Huffman / Prefix codes. RDF List serialization. Meta Resources / Models declarative Encoding, Addressing, Mappings, Transforms (Immutable sequences, dataflow Mapping: Template / Augmentation / Transform functional streams).

Kinds, Signatures. Contents. Contextual metadata. Sets (bitstring cuads). Lattices.

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors (Augmentation) behavior encoded in statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

Interaction Model (Interaction Level):

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

Order / comparisons: tree representation ordered by Context Role class hierarchy, instances hierarchies and aggregation hierarchies. Resources order (IDs). Statements order (Statement IDs). Comparison criteria (choose relevant IDs). ToDo.

Semantic resolution: Query Resource(s) satisfying “criteria” (i.e.: Object(s) for predicate) IDs by IDs resolution pattern:

Query Resources by role in context.

Query Resources by attributes / values.

Query Resources by identity / type.

Context Kind: Functional stream of Context Statements (Occurrences).

Subject Kind: Functional stream of Subject Statements (Occurrences).

Predicate Kind: Functional stream of Predicate Statements (Occurrences).

Object Kind: Functional stream of Object Statements (Occurrences).

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

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.

Messages CRUD / Invocation semantics. Dialog. Prompts.

Encoding: Cons lists. Trees. Huffman / Prefix codes. RDF List serialization. Meta Resources / Models declarative Encoding, Addressing, Mappings, Transforms (Immutable sequences, dataflow Mapping: Template / Augmentation / Transform functional streams).

Kinds, Signatures. Contents. Contextual metadata. Sets (bitstring cuads). Lattices.

(C (S (P (O, Nil))));

(C2 (C (S (P, Nil)));

Encoding / Dimensional example: role in context. X is Y for Z in W.

(W (Z (X (Y))));

(Marriage (Role (Man (Husband))));

(Hour (Minute (1 (60))));

Encoding, Dimensional, Meta Model. Units. Events. Order. Relations. Comparison. Input layers. Augmentation.

**Messages**

Monadic Functional Statement (Resource) wrapper.

Messages: Message semantics (Augmentation: Verbs, CRUD, Behavior) according Message structure / pattern (dialog / prompts).

Augmentation result: Message / interaction layer matching / populated Transform Statement. Template, Mapping, Transform Augmentation Meta Resources.

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.

Persistence:

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.

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

Augmentations: Interaction Model Mappings execution / persistence / retrieval. Reactive model via representation of IDs: Mappings (signatures) dataflow inferred Augmentations.

Persistence: (activation / passivation): IDs / Meta Model / Facets from Interaction Model events (Messages) from Node IO. Interaction Model: Main Model(s) Message IO.

Messages: Saga Activation. Interaction Model (Meta Model). Aggregated (Interaction) Meta Model interactions (performed / inferred / possible) emitted as Model event Messages (Saga pattern). Mappings.

Saga Activation / Passivation populating Node local Quad store / persisting peers via DIDs ([ont.io](http://ont.io)) semantic (resolvable / discoverable) identifiers.

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.

Messages: Dataflow. Subscriptions. Reactive Model. Dynamic subscriptions / bindings. Events publish / subscribe between Model Resource. Mappings.

Resolve Message matching Resource from behavior layers / matching kinds from Model / data layers.

(Kind, SuperKind, Attribute, Value);

(Occurrence, Kind, SuperKind, Attribute);

(Context, Occurrence, Kind, SuperKind); (attributes / links bindings).

(Resource, Context, Occurrence, Kind); State Resource Kind in occurrence context (context / role bindings).

(Statement, Resource, Context, Occurrence); State Resource URIs occurrences / Resource class IDs (classification bindings).

(Interaction, Statement, Resource, Context);

(Action, Interaction, Statement, Resource);

Example: a message composed of a kinds CSPO matches statements “instances” of those specifications (statements whose CSPO have matching kinds). A message with three CSP kinds and a (potentially unknown) object URI retrieves matching resources having that object value into corresponding property kinds. An statement of plain (potentially unknown) URIs instantiates / updates and augments new / known resources added to models and returns an augmentation transform result.

Interaction Model: Context of Messages model for a given interactions session / dialog state. Message invocation requests: Statement(s) building Resource invocation graph with layers matching Message patterns. Layers graph invocation patterns matching from higher to lower layers resources fulfilling higher layers templates. Variables, wildcards, placeholders.

Dialog arguments resolutions example: higher layer Resource / Message request / invocation instantiates in Interaction Transform context corresponding lower layer graph statements to be “populated” to fulfill request. Message IO of “forms” (Messages) inter-peers (originating peer  
acting as “server”) for initial requested peer to “ask” for form elements to be populated (interaction context “dialogs”). Resolution may propagate to other peers (content aware addressing dataflow routes dispatch: P2P resources address encodings, matching forms models requests). Nested interactions.

Explain messages (resource resolution). Grammar. Match model Resource(s). Compound nested CSPO statement contexts defines result behaviors. Message CSPO contexts may define create, retrieve, update or delete operations (passing 'null' for example for resource / statement to be deleted).

Explain transforms (message application). Transform: Resource stream result of Message application over resolved Resource(s)). Input statements: Message(s) / Resource(s) (from input message or to be populated or populated in dialog) and "goal" Message / Resource aggregating a model from Resource MetaGraph with Message / Resource bindings.

**Dataflow**

Interaction Layer declares Events (Augmentations) which have a functional Mapping between its domain (Template) and range (Transform). An Augmentation Context Kind correspond to this Mapping “signature”. Dataflow binds input Message(s) to domain Template by pattern matching and resolving any input Message references (Addressing).

Outputs are resolved by pattern matching with Transform, Message and existing Model data. Augmentations may play the role of “placeholder” Resource(s) which are bound to context aware Augmentations thus rendering Transforms into Model entities (including Mapping Augmentations themselves).

Model declared as Interaction Layer Augmentation(s) (matching Messages) in Interaction Model. Flows.

Transform result adds new Context / Occurrence, merges existing Context Occurrence in Model with new Statements (prompts / embeddings). Dataflow.

Augmentation: Event emits new dataflow Message Transforms from Template Mappings.

Augmentation: Input Templates prompts. Dialog, Meta Resource embedded Templates / Mappings / Augmentations in input Message / output Transform.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

(Template, Mapping, Transform, Model);

(Augmentation, Template, Mapping, Transform);

(Model / Template): class / superclass (intension / extension).

Model declared as Interaction Model Augmentation (matching Messages) in Interaction Model. Flows (levels):

Message

(\*) Model (Augmentation Functor)

Template (Message) : populate / prompt

Mapping (Functor)

Transform (Message) : populate / prompt

(\*) Model (Augmentation Functor)

Message

(\*): Feedback.

Interaction Model: Model Events (Augmentation).

Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output.

Embeddings: Message Match Event Signature. Tempate matching / Transform rendering. Dataflow.

Embeddings: Meta Model Augmentation, Template, Mapping, Transform Meta Resources (input layer). Meta Model Source, Session, Interaction levels.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

Message routing / discovery resolution. Context Kind Signatures.

Models:

URI(s);

OntResource; Merged URI(s) wrapper.

Resource (OntResource CSPO / Contexts hierarchies Monad wrapper);

Message (Resource Monad wrapper); Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

Augmentation : Functor.

Mappings: Declarative IO signatures: Context Kinds Templates / Transforms. Subscriptions / routes. Dataflow.

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors behavior encoding statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

**Augmentation**

Model declared as Interaction Layer Augmentation(s) (matching Messages) in Interaction Model. Flows.

Transform result adds new Context / Occurrence, merges existing Context Occurrence in Model with new Statements (prompts / embeddings). Dataflow.

Augmentation: Event emits new dataflow Message Transforms from Template Mappings.

Augmentation: Input Templates prompts. Dialog, Meta Resource embedded Templates / Mappings / Augmentations in input Message / output Transform.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

(Template, Mapping, Transform, Model);

(Augmentation, Template, Mapping, Transform);

(Model / Template): class / superclass (intension / extension).

Model declared as Interaction Model Augmentation (matching Messages) in Interaction Model. Flows (levels):

Message

(\*) Model (Augmentation Functor)

Template (Message) : populate / prompt

Mapping (Functor)

Transform (Message) : populate / prompt

(\*) Model (Augmentation Functor)

Message

(\*): Feedback.

Interaction Model: Model Events (Augmentation).

Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output.

Embeddings: Message Match Event Signature. Tempate matching / Transform rendering. Dataflow.

Embeddings: Meta Model Augmentation, Template, Mapping, Transform Meta Resources (input layer). Meta Model Source, Session, Interaction levels.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

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.

Interaction Model for Encoding / Addressing (Mapping : Event routes) Dataflow metadata.

Augmentation: Described in Interaction Layer. 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).

**Augmentation**

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.

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).

Models:

URI(s);

OntResource; Merged URI(s) wrapper.

Resource (OntResource CSPO / Contexts hierarchies Monad wrapper);

Message (Resource Monad wrapper); Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

Augmentation : Functor.

Mappings: Declarative IO signatures: Context Kinds Templates / Transforms. Subscriptions / routes. Dataflow.

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors behavior encoding statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

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).

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.

**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:

Protocols:

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

Implementation: Spring / Vert.x.

Spring: Vert.x / APIs Factories. Services.

Core Messaging / Event Bus backend Service Bean.

Persistence: Topic / Subject wrapping ont.io DIDs Saga (Semantic IDs) Messaging pattern.

Core Meta Resource / Meta Model. Mappings. Service Bean.

Core Model Facets / Levels / Layers Functional Service (streams: Augmentation) APIs.

Message: Augmentation (Encoding) request / response. Mapping: routes / contexts (dataflow).

Index, Registry, Naming Hypermedia Service Beans. Backend, Session, Interaction Levels: Functional Service stream APIs Beans.

DOM (Dynamic Object Model) OGM (Object Graph Mapping). Beans API. JAF (JavaBeans Activation Framework). REST / Client OO APIs. Service Bean.

Apache ServiceMix / JBoss Fuse.

Karaf. Bundles.

OSGi wrapper for Spring / Vert.x. declarative services. Event bus. Discovery (Semantic IDs). Camel.

CXF. Endpoints. Servicr Connectors.

ActiveMQ

Camel. Backend Connectors.

ToDo.

**Client APIs**

Message APIs: Augmentation / Dialog Protocol. Connectors. Services.

Hypermedia APIs: Augment, Extend, Declare. REST. Extended Content Type signatures Activation / Dataflow. Services.

Wiki like abstract representation / protocol. Template rendering. Services.

DCI Activation DOM OGM. REST. API Client. Services.

# IDs: Encoding / Addressing

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.

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.

Messages CRUD / Invocation semantics. Dialog. Prompts.

Encoding: Cons lists. Trees. Huffman / Prefix codes. RDF List serialization. Meta Resources / Models declarative Encoding, Addressing, Mappings, Transforms (Immutable sequences, dataflow Mapping: Template / Augmentation / Transform functional streams).

Kinds, Signatures. Contents. Contextual metadata. Sets (bitstring cuads). Lattices.

(C (S (P (O, Nil))));

(C2 (C (S (P, Nil)));

Models:

URI(s);

OntResource; Merged URI(s) wrapper.

Resource (OntResource CSPO / Contexts hierarchies Monad wrapper);

Message (Resource Monad wrapper); Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

Augmentation : Functor.

Mappings: Declarative IO signatures: Context Kinds Templates / Transforms. Subscriptions / routes. Dataflow.

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors behavior encoding statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

Message (Resource Monad wrapper) : Statement; Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

Augmentation : Functor.

Mappings: Immutable Resources Message based Augmentation bindings. Dataflow subscription routes: Signatures / CKs (Augmentation(s) functional streams).

Subject Kind: Subjects stream. Object Kind: Objects stream.

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors (Augmentation) behavior encoded in statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

Interaction Model (Interaction Level):

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

ToDo.

**Encoding:**

IDs:

URI(s);

OntResource; Merged URI(s) wrapper.

OntResource hierarchy: layers statement contexts. Facets DOM, Actor / Role.

Resource (OntResource Context Roles hierarchies Monad wrapper);

Statement : Resource quad, Resource.

Message (Resource Monad wrapper); Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

IDs:

A: OntResource.

B: CSPO Role.

C: Statement : OntResource Occurrence.

D: Kind CSPO Instances.

E: Class : Kind CSPO Classes.

F: ContextStatement : Context Role.

Meta Model:

A: (Resource, ?, ?, ?);

B: (Role, Resource, ?, ?);

C: (Statement, Role, Resource, ?);

D: (Kind, Statement, Role, Resource); Data (Resource Kind).

E: (Class, Kind, Statement, Role); Schema (Role Class)

F:.(ContextStatement, Class, Kind, Statement); Interaction (Statement ContextStatement).

ID: (F (E (D (C (B (A, Nil))))));

Cons lists. Binary Trees. Huffman / Prefix codes. RDF List serialization. Meta Resources / Models declarative statements Encoding, Addressing.

Kinds, Signatures. Contents. Contextual metadata.Lattices. Roles.Sets (bitstring cuads). Definitions (elements). Operations. Rules. Categories. Groups.

Statement: (ID (ID (ID (ID, Nil))));

(C (S (P (O, Nil))));

IDs, Meta Model, Interaction Model (Level), Session (Level), Backend (Level), Facets features:

Augmentations: Interaction Model Mappings execution / persistence / retrieval. Reactive model via representation of IDs: Mappings (signatures) dataflow inferred Augmentations.

Persistence: (activation / passivation): IDs / Meta Model / Facets from Interaction Model events (Messages) from Node IO. Interaction Model: Main Model(s) Message IO.

Order / comparisons: tree representation ordered by Context Role class hierarchy, instances hierarchies and aggregation hierarchies. Resources order (IDs). Statements order (Statement IDs). Comparison criteria (choose relevant IDs). ToDo.

Semantic resolution: Query Resource(s) satisfying “criteria” (i.e.: Object(s) for predicate) IDs by IDs resolution pattern:

Query Resources by role in context.

Query Resources by attributes / values.

Query Resources by identity / type.

Message (Resource Monad wrapper) : Statement; Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

Augmentation : Functor.

Mappings: Immutable Resources Message based Augmentation bindings. Dataflow subscription routes: Signatures / CKs (Augmentation(s) functional streams).

Subject Kind: Subjects stream. Object Kind: Objects stream.

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors (Augmentation) behavior encoded in statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

Interaction Model (Interaction Level):

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

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

Messages: Saga Activation. Interaction Model (Meta Model). Aggregated (Interaction) Meta Model interactions (performed / inferred / possible) emitted as Model event Messages (Saga pattern). Mappings.

**Meta Model:**

URI(s);

OntResource; Merged URI(s) wrapper.

OntResource hierarchy: layers statement contexts. Facets DOM, Actor / Role.

Resource (OntResource Context Roles hierarchies Monad wrapper);

Resource (OntResource CSPO / Contexts hierarchies Monad wrapper);

Role (Model CSPO Context Roles hierarchies type classes) : Resource;

Statement (Resource, Resource, Resource, Resource) : Resource;

Kind (Statement\*) : Resource;

Class (Kind\*) : Resource;

Context (Class\*) : Resource;

Hierarchy: class (Object / Value) as superclass Context.

Object: class (extension);

ContextStatement: super class (intention); Context Role.

(Resource, ?, ?, ?);

(Role, Resource, ?, ?);

(Statement, Role, Resource, ?);

(Kind, Statement, Role, Resource); Data (Resource Kind).

(Class, Kind, Statement, Role); Schema (Role Class)

(ContextStatement, Class, Kind, Statement); Interaction (Statement ContextStatement).

State Facet / Layer / Level / Augmentation / Model Resource Mappings.

Functional API: Message IO. Mappings.

Interaction Model (Interaction Level):

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

State order (in context class hierarchies axes), comparison relations, iterations, flow, events, causal relations, units, enums, equivalence, etc.

Data order: Resource Kind hierarchies.

Schema order: Role Class hierarchies.

Interaction order: Statement Context hierarchies.

## Models / Meta Model IDs

IDs:

URI(s);

OntResource; Merged URI(s) wrapper.

OntResource hierarchy: layers statement contexts. Facets DOM, Actor / Role.

Resource (OntResource Context Roles hierarchies Monad wrapper);

Statement : Resource quad, Resource.

Message (Resource Monad wrapper); Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

IDs:

A: OntResource.

B: CSPO Role.

C: Statement : OntResource Occurrence.

D: Kind CSPO Instances.

E: Class : Kind CSPO Classes.

F: ContextStatement : Context Role.

Meta Model:

A: (Resource, ?, ?, ?);

B: (Role, Resource, ?, ?);

C: (Statement, Role, Resource, ?);

D: (Kind, Statement, Role, Resource); Data (Resource Kind).

E: (Class, Kind, Statement, Role); Schema (Role Class)

F:.(ContextStatement, Class, Kind, Statement); Interaction (Statement ContextStatement).

ID: (F (E (D (C (B (A, Nil))))));

Cons lists. Binary Trees. Huffman / Prefix codes. RDF List serialization. Meta Resources / Models declarative statements Encoding, Addressing.

Kinds, Signatures. Contents. Contextual metadata.Lattices. Roles.Sets (bitstring cuads). Definitions (elements). Operations. Rules. Categories. Groups.

Statement: (ID (ID (ID (ID, Nil))));

(C (S (P (O, Nil))));

ToDo.

## Resolution: graph criteria / query

ToDo.

## Sorting / Order / Comparisons

Order / comparisons: tree representation ordered by Context Role class hierarchy, instances hierarchies and aggregation hierarchies. Resources order (IDs). Statements order (Statement IDs). Comparison criteria (choose relevant IDs). ToDo.

Semantic resolution: Query Resource(s) satisfying “criteria” (i.e.: Object(s) for predicate) IDs by IDs resolution pattern:

Query Resources by role in context.

Query Resources by attributes / values.

Query Resources by identity / type.

ToDo.

# Model Functional APIs

## Resource Monad

Monadic wrapper for which Augmentation (Functor Events) are declared into Interaction Model. Model(s) themselves are Augmrntation(s). Augmentation Statement Context Kind defines Event “signature”: Resource input / output Event domain / range. Output from an Event application (Transform) may feed back Model triggering further events (Dataflow). Augmentation Template, Mapping and Transform may behave as placeholder for Dataflow rendering of Meta Models.

ToDo.

## Reactive / Events (Resource Monad)

Model Resources react to events according Message matching event “signature”.

ToDo.

## Meta Resources

Declare Meta Model Meta Resources as Interaction Model Augmentations, Mappings, Templates and Transforms. Model(s) (Facets) are “root” Augmentations over input Message Statements. Further Dataflow and Meta Model Interaction Resources embeddings shapes Meta Model instances.

ToDo.

## Meta Model

Meta Resources built Model which instantiates Interaction Model statements into layered Facets Models.

ToDo.

## Streams (Context, Kinds, etc.)

Context Kind: Functional stream of Context Statements (Occurrences).

Subject Kind: Functional stream of Subject Statements (Occurrences).

Predicate Kind: Functional stream of Predicate Statements (Occurrences).

Object Kind: Functional stream of Object Statements (Occurrences).

ToDo.

# Messages

Messages: Mappings. Meta Resources / Model Message based Model interactions (Subscriptions / Mappings).

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.

Mappings: Declarative IO signatures: Context Kinds Templates / Transforms. Subscriptions / routes. Dataflow.

Messages: Mappings. Meta Resources / Model Message based Model interactions (Mappings : 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 / content alignments). 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.

ToDo.

## Message Monad

Encode Model Events Interactions (Augmentation) inputs (Template) and outputs (Transform) wrapping corresponding Resource(s).

ToDo.

## Reactive / Events (Message Monad)

(Augmentation, Template, Mapping, Transform);

Model declared as Interaction Model Augmentation (matching Messages) in Interaction Model. Flows:

Message

Model (Functor)

Augmentation (Addressable Interaction)

Template (Message)

Mapping (Functor)

Transform (Message)

Model (Functor)

Message

Interaction Model: Model Events (Augmentation).

Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output.

Embeddings: Message Match Event Signature. Tempate matching / Transform rendering. Dataflow.

Embeddings: Meta Model Augmentation, Template, Mapping, Transform Meta Resources (input layer). Meta Model Source, Session, Interaction levels.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

ToDo.

## Model Functional APIs

Invoke Augmentation over Resource Message matching Event signature.

ToDo.

## IDs: Encoding / Addressing

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors behavior encoding statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

IDs: Addressing / Encoding. Semantic (signature, contents, context) resolvable / discoverable identifiers.

ToDo.

## Persistence

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.

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

Augmentations: Interaction Model Mappings execution / persistence / retrieval. Reactive model via representation of IDs: Mappings (signatures) dataflow inferred Augmentations.

Persistence: (activation / passivation): IDs / Meta Model / Facets from Interaction Model events (Messages) from Node IO. Interaction Model: Main Model(s) Message IO.

Messages: Saga Activation. Interaction Model (Meta Model). Aggregated (Interaction) Meta Model interactions (performed / inferred / possible) emitted as Model event Messages (Saga pattern). Mappings.

Saga Activation / Passivation populating Node local Quad store / persisting peers via DIDs ([ont.io](http://ont.io)) semantic (resolvable / discoverable) identifiers.

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.

ToDo.

# Augmentation

Interaction Model Event. Matches Message signature (domain Template / range Transform) performing Mapping. Dataflow: Transform output matches another Event signature. Embedding: OntResource augmented with new referenced aligned / matched Model entity.

Augmentations defined as declarative Mappings in Interaction Model encoding Context (layer) inputs matching signatures and augments current / previous layer emmiting mapping transforms.

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.

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.

## Reactive / Events (Functors)

Augmentations: matching Events Functors 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.

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.

ToDo.

## Interaction Model

(Augmentation, Template, Mapping, Transform);

Model declared as Interaction Model Augmentation (matching Messages) in Interaction Model. Flows:

Message

Model (Functor)

Augmentation (Addressable Interaction)

Template (Message)

Mapping (Functor)

Transform (Message)

Model (Functor)

Message

Interaction Model: Model Events (Augmentation).

Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output.

Embeddings: Message Match Event Signature. Tempate matching / Transform rendering. Dataflow.

Embeddings: Meta Model Augmentation, Template, Mapping, Transform Meta Resources (input layer). Meta Model Source, Session, Interaction levels.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

ToDo.

## Message

Augmentation Event input (Template) / output (Transform) declaration / instance (Mapping).

ToDo.

## Augmentation

ToDo.

## Template

ToDo.

## Mapping

Mappings / Augmentation Context Kind.

ToDo.

## Transform

Augmentation Mapping range declaration / result instance (Transform Message).

ToDo.

## IDs: Encoding / Addressing

Contextual / patterns embedding metadata / resolution. Augmentation occurrences Dataflow sources / context sync.

ToDo.

# Dataflow

(Augmentation, Template, Mapping, Transform);

Model declared as Interaction Model Augmentation (matching Messages) in Interaction Model. Flows:

Message

Model (Functor)

Augmentation (Addressable Interaction)

Template (Message)

Mapping (Functor)

Transform (Message)

Model (Functor)

Message

Interaction Model: Model Events (Augmentation).

Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output.

Embeddings: Message Match Event Signature. Tempate matching / Transform rendering. Dataflow.

Embeddings: Meta Model Augmentation, Template, Mapping, Transform Meta Resources (input layer). Meta Model Source, Session, Interaction levels.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

ToDo.

## Reactive / Events (Functors)

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

## Interaction Model

(Augmentation, Template, Mapping, Transform);

## Messages

ToDo.

## Augmentation

ToDo.

## Template

ToDo.

## Mapping

ToDo.

## Transform

ToDo.

## IDs: Encoding / Addressing

Contextual / patterns embedding metadata / resolution. Augmentation occurrences Dataflow sources / context sync.

# Ontology Matching

ToDo.

## Ontology Merge

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.

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.

ToDo.

## Alignments (data / schema / behavior).

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).

ToDo.

## IDs: Encoding / Addressing

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.

ToDo.

# 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:

Protocols:

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

Implementation: Spring / Vert.x.

Spring: Vert.x / APIs Factories. Services.

Core Messaging / Event Bus backend Service Bean.

Persistence: Topic / Subject wrapping ont.io DIDs Saga (Semantic IDs) Messaging pattern.

Core Meta Resource / Meta Model. Mappings. Service Bean.

Core Model Facets / Levels / Layers Functional Service (streams: Augmentation) APIs.

Message: Augmentation (Encoding) request / response. Mapping: routes / contexts (dataflow).

Index, Registry, Naming Hypermedia Service Beans. Backend, Session, Interaction Levels: Functional Service stream APIs Beans.

DOM (Dynamic Object Model) OGM (Object Graph Mapping). Beans API. JAF (JavaBeans Activation Framework). REST / Client OO APIs. Service Bean.

Apache ServiceMix / JBoss Fuse.

Karaf. Bundles.

OSGi wrapper for Spring / Vert.x. declarative services. Event bus. Discovery (Semantic IDs). Camel.

CXF. Endpoints. Servicr Connectors.

ActiveMQ

Camel. Backend Connectors.

ToDo.

# Client APIs / Connectors

## Services

Message APIs: Augmentation / Dialog Protocol. Connectors. Services.

Hypermedia APIs: Augment, Extend, Declare. REST. Extended Content Type signatures Activation / Dataflow. Services.

Wiki like abstract representation / protocol. Template rendering. Services.

DCI Activation DOM OGM. REST. API Client. Services.

ToDo.

URIs, Resource, Contexts Functional APIs

Services:

Registry.

Naming.

Index.

Connectors (URIs):

JDBC.

Meta Model:

URI;

Resource (URI\*);

Role (Model CSPO hierarchies) : Resource;

Statement (Resource, Resource, Resource, Resource) : Resource;

Kind (Statement\*) : Resource;

Class (Kind\*) : Resource;

Context (Class\*) : Resource;

Hierarchy: class (Object / Value) as superclass Context.

Object: class (extension);

Context: super class (intention);

(Kind, Statement, Role, Resource); Data (Resource Kind).

(Class, Kind, Statement, Role); Schema (Role Class)

(Context, Class, Kind, Statement); Interaction (Statement Context).

State Facet / Layer / Level / Augmentation / Model Resource Mappings.

Functional API: Message IO. Mappings.

State order (in context class hierarchies axes), comparison relations, iterations, flow, events, causal relations, units, enums, equivalence, etc.

Data order: Resource Kind hierarchies.

Schema order: Role Class hierarchies.

Interaction order: Statement Context hierarchies.

Resource / Message Monad Events: Augmentations. Mapping: Endpoint. Events: Implement Message / Resource / URIs Protocols.

Services. Connectors. URIs APIs.

Endpoints (Events Mapping) messaging interface.

ToDo.

**Services**

Services: Augmentation interaction layer and their corresponding Model(s).

Interaction Model Event. Matches Message signature (domain Template / range Transform) performing Mapping. Dataflow: Transform output matches another Event signature. Embedding: OntResource augmented with new referenced aligned / matched Model entity.

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, Interaction 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.

Service URIs:

Service URIs: Context Kind (inputs / outputs domain / range). Example: predictions, classification, clustering, regression. Index / Naming / Registry "contexts" (facets).

Extended content types activations on domain / range (verbs, augmentations). Example: image, face, coords.

Functional Resource Model / Context / Attributes / Kind design / implementation. Serialization (Encoding / Models). Signatures. Reactive. Augmentation. DOM, Actor / Context / Role. APIs: Augmentation. Meta Resources.

Index

Naming

Registry

Service (URIs APIs). Index. Naming. Registry. Custom (signatures : Context Kind).

**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.

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.

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).

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. Dataflow.

Layered models semantic infrastructure for integration of heterogeneous backends (meta models).

Alignments Augmentations:  
  
Activation: type inference : classification (determine class / metaclass / roles for entity attributes and values).  
  
Activation infer attributes / relations : clustering (from multiple occurrences of same entity in diverse data sources).  
  
Aggregation: infer roles in contexts: regression (Person class in Employment interaction : Developer role).  
  
Integration of addressable resources. Reactive I/O (sync back ends). Content type driven semantic augmentation / annotations.  
  
Integrated view. Navigate contexts, data, interactions. APIs. Dimensional views annotations (analysis / mining).  
  
Augmentation of distributed resources. Annotations (Semantic / ML). API for resource / schema / interactions exploration / protocol for message based API "dialogs" execution. HAL (Hypertext Application Language), OData (REST) like interfaces.  
  
Example: Google Drive / Google Knowledge Graph APIs Augmented with ML / Semantic intelligence tailored for specific domains / application kinds.

**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:

Protocols:

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

Implementation: Spring / Vert.x.

Spring: Vert.x / APIs Factories. Services.

Core Messaging / Event Bus backend Service Bean.

Persistence: Topic / Subject wrapping ont.io DIDs Saga (Semantic IDs) Messaging pattern.

Core Meta Resource / Meta Model. Mappings. Service Bean.

Core Model Facets / Levels / Layers Functional Service (streams: Augmentation) APIs.

Message: Augmentation (Encoding) request / response. Mapping: routes / contexts (dataflow).

Index, Registry, Naming Hypermedia Service Beans. Backend, Session, Interaction Levels: Functional Service stream APIs Beans.

DOM (Dynamic Object Model) OGM (Object Graph Mapping). Beans API. JAF (JavaBeans Activation Framework). REST / Client OO APIs. Service Bean.

Apache ServiceMix / JBoss Fuse.

Karaf. Bundles.

OSGi wrapper for Spring / Vert.x. declarative services. Event bus. Discovery (Semantic IDs). Camel.

CXF. Endpoints. Servicr Connectors.

ActiveMQ

Camel. Backend Connectors.

ToDo.

**Client APIs**

Message APIs: Augmentation / Dialog Protocol. Connectors. Services.

Hypermedia APIs: Augment, Extend, Declare. REST. Extended Content Type signatures Activation / Dataflow. Services.

Wiki like abstract representation / protocol. Template rendering. Services.

DCI Activation DOM OGM. REST. API Client. Services.

ToDo.

**Deployment**

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

Deployable entity: Node. Publish / Subscribe (Message) signatures (interface). Augmentation / Mappings Interaction Model (Runtime). Models, Facets, Services, etc. ToDo.

**ToDo**

Interaction Layer in Meta Model. Rendering (Meta Model Message IO Augmentation): Aggregation, Alignment, Activation of Meta Model / Model Facets (sync / merge). Ontology Matching.

Encoding / Meta Model:

Resource;

Role (CSPO);

Statement (Resource, Resource, Resource, Resource) : Resource;

Kind (Statement\*) : Resource;

Class (Kind\*) : Resource;

Context (Class\*) : Resource;

Hierarchies:

Context: class (extension);

Object: super class (intension);

Meta Model Layers:

(Template, Mapping, Transform, Model);

(Augmentation, Template, Mapping, Transform);

A: (Resource, Augmentation, Template, Mapping);

B: (Role, Resource, Augmentation, Template);

C: (Statement, Role, Resource, Augmentation);

D: (Kind, Statement, Role, Resource);

E: (Class, Kind, Statement, Role);

F:.(Context, Class, Kind, Statement);

Contexts (example):

ID: (F (E (D (C (B (A, Nil))))));

ID: (F:E (D:C (B:A, Nil))))));

Contents (example: F occurrence contents):

F: (F:E (D:C (B:A, Nil))))));

IDs: DIDs. Resource Addresses. Occurrences Context / Content Encoding. Template / Transform Mapping Augmentation routes discovery (patterns).

Encode IDs: Contextual occurrence metadata. Declarative (Meta) Resources. Enable algorithmic Ontology Matching, Augmentation / Rendering and Service encoded Resource resolution.

Encoding: XML / XSL. IDs: nested lists, XSL to / from RDF Quads. Template matching. Ontology Matching / Rendering / Augmentation encoded mapping transform Augmentation algorithms.

Functional API:

Resource<OntResource>;

Message<Resource>;

Template<Message, Kind>;

Transform<Message, Kind>;

Mapping : Functor<Template, Transform>;

Render possible Mapping : Augmentation signature (Context Kind).

Mapping Kinds: Template Resource Kind mapped to Transform Resource Kind. Dataflow. Monadic map / unit /bind.

Message: dialog prompts. CSPO Kind Mapping. Resolve Mapping according contexts. Posible Augmentations: variables, placeholders, expressions. Perform possible Augmentations (CDI) Dataflow from Meta Model Interaction Layer rendered input Messages. Monadic map / unit / bind.

Order: same Augmentations Context, concatenated Template(s) / Transform(s). Hierarchies: Context: class (extension); Object: super class (intension);

Encoding / Dimensional example: role in context. X is Y for Z in W.

(W (Z (X (Y))));

Examples:

(Marriage (Role (Man (Husband))));

(Hour (Minute (1 (60))));

Encoding, Functional, Semiotic, Dimensional, Meta Model. Units. Events. Order. Relations. Comparison. Input layers. Augmentation.

Ontology Matching:

Facets / Levels (Schema):

Functional: (Class, Role, Entity, Statement);

Semiotic: (Context, Sign, Concept, Object);

Dimensional: (Dimension, Unit, Measure, Value);

Werk: Deployment, Platforms. Messaging. Persistence. RDF Backend. Functional Message Driven Model APIs.

Reactive Dataflow (Resource Interaction Layer: Message Augmentation). Functional content Activation (Context Kinds: domain / range; order).

Services Endpoints: APIs (DCI Dialog). OGM. OData. HAL. Index. Naming. Registry. Connectors.

DIDs: Endpoints, Ontology Matched URIs. P2P. Persistence.

Encoding (Dataflow): Monads / Functional / MapReduce (Augmentation Transform Template Mapping / Ontology Matching map / reduce events: Properties dot notation graph encoding) IO. Connectors / Persistence streams.

Architecture. Services: Model, API, Endpoints (Messaging, Connectors: Services). Dataflow: Services bindings (Service Model).

Implementation:

Architecture. Configuration: Bus, Services (Model, APIs), Connectors (Interaction APIs, Implementation). Dataflow: Services bindings signatures / interactions (Service Model APIs).

Services: Model, APIs, Reactive Dataflow Protocol (Bus Dialog Services Model Messaging).

Connectors: Functional Services APIs. Reactive Services APIs I/O. Interactions integration (adapters) APIs. Levels abstraction.

Configuration: Main peer process. Spring, Vert.x, ServiceMix, JMS / CDI, JCA, JAF, Rx, others. Dynamic peer Configuration: Services, Connectors. Runtime bindings.

Bus: Resource publish / subscribe. Topics, Queues.

Bus: Signatures. Service Model exported domain / range (Services interface).

Services: Bus Resource publish / resolve (Service discovery). Message Activation performs Services Augmentations (knowledge / behavior transforms).

Services (Dataflow signature interfaces from Service Model): Source (poll / feed) Service. Sink (output / dest). Processor (consumer / producer).

Connectors: Services Resource I/O events translation (routing). Integration gateways / facades. Protocols / Representations. Hypermedia objects activation. Resource descriptions in types and contexts.

Connectors: Persistence (DIDs), Reasoning / Inference, Integration (DBs, services, apps), Clients (HAL, OData, OGM) etc. Extended headers (referrer, context) / content types.

Bus, Application, Configuration, Service, Connector: Beans.

Bus: JMS. Dataflow (aggregation): Address, Event, Stream.

Application (Jersey): CDI, JMS, JavaRx: Functional bindings / helpers. Model (Bus persisted).

Configuration: Application, Bus. Reactive Dataflow Services mappings / subscriptions (Address signatures / mappers / transforms).

Service: Configuration, Events. Reactive Dataflow Connector mappings / subscriptions (Event signatures / mappers / transforms).

Connector: Service, Streams. Reactive Dataflow Endpoints / APIs mappings / subscriptions (Stream signatures / mappers / transforms).

Services examples: Ontology Matching, ML / Extended Content type activation meta Resources, Domains (Levels). Routes (rendezvous peers bindings).

Connectors example: REST / HAL, OData, Solid, Tryton, GNUHealth. Domain Model(s).

Application Model (MDM / Alignments / Matching: distributed Services Models state orchestration).

Configuration.

Bus: events addresses / messages / dataflows declaratively stated in Models Interaction Layer. Application Configuration Service Connector composition.

Containerized Service APIs. Platforms Protocol bindings (Bus integration mappings). Services Models (Domains).

Containerized Platforms (Java, NodeJS, PHP, etc.) Services. Service Model.

Service / Connector Endpoints integration mappings. Interaction Model, Levels, Gestures. Configurations.

Connectors:

ISO 15926 Connector.

SPARQL Connector.

ISO 13250 DM / RM Connector (Meta / Augmented Content Types / JAF Activation Protocol Connector).

REST / HAL Connector.

Solid Connector.

Web / JavaScript Connector.

OData Connector.

JCA Connector.

JBoss Teiid / Apache Metamodel / JDBC Connector.

JMS Connector.

Others.

Protocol: Distributed data. Consolidation. Model driven. Dimensional. Timestamp. Axes. Facets. Facts. Ontology Matching. Predict. Regress. Classify. Consolidate. Distributed Inference and Aggregation / Addressing of Dataflow Routes (Futures: Augmentation Mapping Template / Transforms "slots" / Roles alignments). Aggregate matching resources (Resource URIs / Actors). Encode dimensional relationships (Models).

Resource aggregates URIs matching (occurrence of concept in role in context). Concept equivalences matching. Occurrence: Role / Context / Player aggregation. Player wrapping reified Concept (Resource).

**Features**

Graph encoding of data / schema / behavior. Dimensional / Grammar annotations. MetaGraph: augmentation / transforms (Messages). Features.

Parallel distributed graphs models augmentation / transforms synchronization (Messages). Event sourcing (distributed inferences). P2P / DIDs.

Augmentation. Ontology matching. Hypermedia augmentation protocol. Browser / Client APIs.

URIs API for annotating network retrievable resources metadata. Content type / model driven augmentations / activations (models features / outputs). Subject attributes / values. Occurrences contexts / roles. Paths, pointers, locators. Example: annotate document URIs (parts, sections, mentions), annotate images URI (whole image description, coords: classes, individuals), annotate DB, table, row, column, value URIs, annotate / describe service / APIs URIs. Hypermedia protocol composable with other (described / annotated) APIs / resources. Example: Drive APIs.

Deployable entity: Node. Publish / Subscribe signatures (interface). Augmentation / Mappings Interaction Model (Runtime). Models, Facets, Services, etc. ToDo.

**To Do:**

* Resource / Message Monad Events: Augmentations. Mapping: Endpoint. Events: Implement Message / Resource / URIs Protocols.
* Services. Connectors. URIs APIs.
* Endpoints (Events Mapping) messaging interface.
* Meta Resource / Meta Model:
* Model, URIs, Resource, Contexts Functional APIs. Meta Model / Resources encoding. Mappings.
* 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.
* IDs: Addressing / Encoding. Semantic (signature, contents, context) resolvable / discoverable identifiers.
* Meta Resource / Models / Messages: IDs / Encoding / Addressing formats. Ontology matching and Template / Augmentation / Transform enrichment (alignments), transforms (functors), materialization (model updates) via Mappings (events) and Meta Resource / Model Encoded Resource declarations (enrich / align, transform, updates algorithms: Encodings).
* 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.
* 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 (Mappings : 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 / content alignments). 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.
* Implementation:
* JavaScript (browser) / NodeJS Core.
* Endpoints. Connectors (OData, HAL, OGM, Spring, ServiceMix / Fuse).
* Client APIs:
* Browser (JavaScript) / NodeJS / Connectors.
* Applications. Use Cases.
* Levels. Gestures / Actions (Services / UX). Rendering (REST APIs / Dynamic UX).

**Implementation:**

Definitions: Quads, contexts, Kinds, Grammar, etc.

Implementation: Runtime. Architecture. Components. Patterns. Models. Messages. Augmentation. Events dispatch. Message aggregation / instantiation / resolution / application. Backends. Services.

Resources inputs / outputs: Augmentation, Protocol, Browser. Message addressing / resolution / application.

Component: Models (data). Source: augmented input statements. APIs (Model).

Component: Messages (contexts). Source: augmented models templates (Grammar). APIs (Model).

Component: Transforms (interactions): Source: input statements case matching Message inputs. Returns / materialize results. APIs (Model).

Core Model API: Augmented (Aligned, Activated, Aggregated inputs matching model context messages) IO. Resource MetaGraph. Dimensional model. Grammar. Model repository. Backend. API.

Core API: Model, URI, Resource, Statement, Kind.

Message: Context Model API. Input statements: Model Grammar. Augmented IO by interaction transforms of applied matching Message with model statements inputs. Context of core models instances. API.

Transform: Interaction Model API. Input statements: Transform request invocation specification. Functional application of Message(s) over Resource(s): Transform (streams). Augmented IO: Requested Transform which applied augments resulting responses (dialog arguments resolutions). Context of context model instances. Reactive / streams API.

Message Transform (interaction result): matches request context specification built upon Resources / Messages (TransformBuilder). Resolve state / dialog session graph. Returns observable stream. Dataflow (chaining). Operations (over streams).

Transform request invocation specifications: means to interact with underlying contexts models (CRUD, domains behavior). Transforms result from applicating Message(s) over Resource(s). Sending a Message Resource to a given interaction context initiates a “dialog” in which to “populate” target Resource(s) and Resource arguments. Each dialog “step” renders resources / layers streams of requested arguments (server “queries” clients) or resources / layers streams of response augmented Resource(s).

Message IO encoding components kinds:

Data: Assertion (statement / entity).

Schema: Type (kind / class).

Behavior: Interaction (flows / behaviors).

Specification resolves to query / create / update / delete according interaction contexts. Messages models determines “possible” messages according models grammars. Interaction specification (statement / graph / dialog) may have any message encoding components in corresponding statement roles. For each behavior, flow, class, kind, entity, statement in input request, transforms matches those components by applying messages into model resources (grammar) matched into interaction model (binding subsecuent roles by dialogs). New (potentially unknown) resources are added and augmented into the graph. Augmented resource events emited from transform streams.

Example: a message composed of a kinds CSPO matches statements “instances” of those specifications (statements whose CSPO have matching kinds). A message with three CSP kinds and a (potentially unknown) object URI retrieves matching resources having that object value into corresponding property kinds. An statement of plain (potentially unknown) URIs instantiates / updates and augments new / known resources added to models and returns an augmentation transform result.

Interaction Model: Context of Messages model for a given interactions session / dialog state. Message invocation requests: Statement(s) building Resource invocation graph with layers matching Message patterns. Layers graph invocation patterns matching from higher to lower layers resources fulfilling higher layers templates. Variables, wildcards, placeholders.

Dialog arguments resolutions example: higher layer Resource / Message request / invocation instantiates in Interaction Transform context corresponding lower layer graph statements to be “populated” to fulfill request. Message IO of “forms” (Messages) inter-peers (originating peer acting as “server”) for inititial requested peer to “ask” for form elements to be populated (interaction context “dialogs”). Resolution may propagate to other peers (content aware addressing dataflow routes dispatch: P2P resources address encodings, matching forms models requests). Nested interactions.

For input Resource(s) (Model reactive / async IO APIs):

. Create / retrieve Model

. Create / retrieve Context Message(s)

. Create / retrieve Message(s) Interactions

. Bind Interaction Message Resource(s)

. Perform Message transform. Materialize results. Message application rules: upper / domain ontology selectors (closest matching role in hierarchies), context alignments.

Match request statement / graph with model via context in interaction (algorithm: addressing, encoding, interaction model upper bindings / alignments). Resource MetaGraph. Reified model resources (Resource, Statement, Kinds, CSPO, etc.).

Apply subsecuent transforms in interaction context (referrer context, get classes playing entity role, get behavior flows, browse / navigate streams). Context, variables, wildcards, placeholders.

Services: distributed addressing / resolution, reactive distributed event bus: streams / contracts, index, naming, registry.

Discovery: All model kinds are browseable / discoverable.

Determine class (reified layers contexts) hierarchies:

(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model resources, statements, kinds.

Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

Models definition: data (Statement, Entity), schema / contex (Role, Class), interactions / behavior (Flow, Behavior).

Kinds / Roles:

Grammar: kinds layers aggregation (CSPO layers Kinds).

Layers: Roles (Models metaclass context resources).

Reified Kind: (Kind, Occurrence, Attribute, Value);

Grammar input set model specificatíon (Statement layer kinds).

Dimensional input set model specificatíon (from Statement layer, ordered SPOs: order criteria, comparisons. Kinds / class / occurrence / instance order criteria?). Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements).

MetaGraph (resolution). Dimensional / Grammar alignments / annotations.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

///

**Purpose driven hypermedia activation:**

Protocols / Services / Clients: Context interaction sessions (state flows).

Content type activation. Messages / gestures. Rules (commands / verbs).

Browser referring context (Work, Peter, Employee).

Models ‘plug’ into Runtime augmenting its capabilities via standard extension APIs (added features / knowledge).

Models ‘modules’: parsing modules declarative descriptions. Augment, link instance data.

Upper aligned ontology plugins / blueprints.

Resource URIs specialized implementations for different connectors / endpoints and content types (DB / OData, REST / HAL, etc.). Feature Resources backends (i.e.: URI for DB interaction).

Purposes: Metamodel declarative goal statement. Fulfill flows (templates / forms: Messages).

Goal: P2P service that connects to services / endpoints (DB, REST, etc.), homogenizes them and exposes an API by which (augmented) knowledge of an stated entity is returned in response (protocol that entails queries / CRUD, object navigation in message / session state contexts). Peer shares / syncs with other peers.

Goal: Intermediate API (HAL for example) aggregating previous objects knowledge (DCI, DOM, OGM, MVC)

Goal: Semantic Browser. Homogenize diverse domains. Query examples. Search session history. Referrer semantics. Collected items in goals roles. Create session purpose document. Link to / from any addressable resource in context / role. Annotate source / destination context roles, attributes and schema.

///

**Resource Activation Protocol**

Annotate, link, browse resources instances, classes, metaclasses, occurrencesin roles in contexts. Services / clients: endpoints: Virtualization (wrapper protocols).

Semantically annotated content types: image/png;face, text/xml;faceImgCoords. RDF schemas describing content, attributes, links in context / target roles.

///

**Messages Metamodel (Context Model):**

Explain models (resources, statements, kinds).

Explain layers / aggregation.

Explain messages (resource resolution). Grammar. Match model Resource(s). Compound nested CSPO statement contexts defines result behaviors. Message CSPO contexts may define create, retrieve, update or delete operations (passing 'null' for example for resource / statement to be deleted).

Explain transforms (message appplication). Transform: Resource stream result of Message application over resolved Resource(s)). Input statements: Message(s) / Resource(s) (from input message or to be populated or populated in dialog) and "goal" Message / Resource aggregating a model from Resource MetaGraph with Message / Resource bindings.

Message types (Augmentation: onto / domains):

Attribute / Link (data):

. Alignment: Augment / infer Attribute / Link.

Class / ID (schema):

. Activation: Augment / infer Kind, Class.

Role / Context (behavior):

. Aggregation: Augment / infer Role / Context.

Runtime / Resources / Messages: Core (upper / onto) Resources, Messages, Transforms. Reified entities (CSPO, Kind, SubjectKind, etc.). Match cases in messages.

Core (upper / onto) Messages: Getters, setters, nav, etc.

Domain Messages: raiseSal: setSal(sal \* increment); promotion: setPosition.

Event sourcing / tracking: married -> marriage occurred.

Resource.flatMap(messageInst::apply) : Resource.

Dataflow: Messages hierarchy. Aggregate contexts from coarse to fine grained transforms (raiseSal -> setAttr).

data <-> schema <-> behavior.

Message dispatch, input statements resolve to applicable messages from switch from behavior to data layer invoking async microservice. Message case matching may involve entering and leaving data, schema and behavior paths if aggregated contexts matches more than one message. Visitor.

Message: functor (monadic transform) : Resource<T> -> R, T, R : URIs (hierarchies, models, semantic content types). Available verbs / flows / navigation (browse models, state of application returned from materialized models). Parameterized functions (partial applications) into Messages metamodel resources. Contexts (dataflow). Execution graph.

Alignment Message: Resource -> Statements (attributes, values).

Activation Message: Statement -> Kind, Class.

Aggregation Message: Statement -> Statement (next layer).

Subscriptions declarations / definitions. Applyied on streams activations (transforms, executions resource parameterized partial contexts).

Messages metamodel: functor declarations partially defined over metamodels resource (T) defining transforms into (R) over appplication (flatMap) over / into (S). Messages inferred / aligned, activated, aggregated according base message transofrms resources. Messages inferred from models / layers. TBD.

Functors <T, R> -> Resource<R>

Form / Template describing (reified as a Resource in a context model) declaratively subscriptions and actual exchange capabilities (datflow). Mappings, Transforms.

Processor which acts upon Resource events. Materialize results.

Specify declaratively augmentations by means of messages.

Upper onto / domain aggregated messages.

Event bus: P2P deployment.

Messages: Monadic applicables over Resource (flatMap).

Base HTTP / Browse (REST) Messages. Custom Messages.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

///

**Addressing: URIs, DIDs URLs. Adddress, content type, representation (URI APIs). Browse / CRUD (DAV).**

Resource<T : URI> monadic hierarchy. Basic hypermedia browse / CRUD (HTTP verbs) bound Message functors compatible for all Resources (REST).

Resource.flatMap(Message::apply) : Observable<Resource> (stream). Composable functions.

Basic Message application (Context Mapping): shift right mapped applied statement resources. Mapped resource context> instance of mesage reified resource context.

Transforms (Message templates):

Resource: Statement

Message: SubjectKind

Transform: Resource

Resource: Statement

Message: ClassLayer

Transform: Statement (class)

Resource: Employee

Message: Position

Transform: Manager

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

Dimensional / Grammar models.

///

**Resource API**

Resource<T : URI> monad. Message functors. Transform reactive extensions.

Transform : Observer / Observable of Resource<T : URI>. Stream. Built upon Resources / Messages (TransformBuilder). Identity and other core transforms (core messages). Stream. flatMap(Message::apply) : Transform<Resource<R : URI>>.

API: Class for layer for model.

API: Class for layer (DOM).

API: Parameterized Resource: layer classes determined by URIs hierarchy, i.e.: Resource<Entity>, Entity : URI.

Base core service URIs (index, naming, registry). URI subclasses implementing / wrapping state for Resource monads offering protocols / addressing / content types / representations facades for services: DBs, WS (REST, SOAP, SPARQL), ML (predictions), etc.

Discovery: All model kinds are browseable / discoverable.

Determine class (reified layers contexts) hierarchies:

(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model resources, statements, kinds.

Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

MetaGraph (resolution). Dimensional alignments / annotations.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

///

**Quads reference model. Kinds. Grammar.**

Declarative means of using RDF quads to state application object models (data, schema and behavior).

Aggregation.

Kinds.

Grammar.

Formalization: Functional / Object API. Reference / Data model. Sets, categories, models.

Subjects: attributes / values, contexts / roles.

(Context, Occurrence, Attribute, Value);

(Context, Sign, Concept, Object);

instance, occurrence, class, metaclass.

Hierarchies: layered quad statements are represented by a class hierarchy which root is the Resource<T> monad. There is a subclass relationship between each layer implementing class and the one of the next layer (Dynamic Object Model).

Quads in the context role of lower layers represents occurrences of context enclosing layer.

Assert class hierarchies, order relation (temporal, causal, containment, etc.) by attrs / vals, set / superset relations. TBD.

Discovery: All model kinds are browseable / discoverable.

Determine class (reified layers contexts) hierarchies:

(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model resources, statements, kinds.

Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

Dimensional / Grammar models.

///

**Metamodels: Graphs (Models)**

Composed of quads semantically aggregated into layers.

Core features provides:

Alignment

Activation

Aggregation

Message / Transform driven specification of Alignment, Activation, Aggregation.

Grammars.

Upper / Dimensional ontology.

Inter models alignments

Services (Endpoint URIs: Resource facades).

Reified model resources (CSPO, Resource, Statement, Kinds, Layers). Augmentation (Alignment, Activation, Aggregation) Messages / Transforms.

Model, Contexts, Interactions IO:

Model: aggregated resource statements.

Context: aggregated model kinds (grammar statements).

Interaction: aggregated model / context bindings.

Inputs: resource statements, resolvable messages. Operation semantics (CRUD, browse, etc.) according input statements layout. Model endpoint. Materializes input resource statements and fully resolved message resource statements from interactions applying Augmentation and matching messages transforms.

Resource flow: input plain RDF URIs statements. Model / Context updates. Transform matches concrete resources.

Resource flow: input message URIs statements. Context / Interaction perform. Transform matches resources in messages context grammar kinds hierarchies.

Outputs: resource statements with possible further resolvable messages (Model IO recursion / dialogs). Interaction queries context / model back for further resolutions. Message transform stream with request message applied plus matching context resolved resources from message.

Model, Contexts, Interactions IO:

Resource, Statement, Kind, Message, Transform.

Subscription, Subscriber, Producer, Consumer, Processor. TBD.

Model: aggregated resource statements model.

Context: aggregated model kinds (grammar statements model).

Interaction: aggregated model / context / dialogs bindings model.

Aggregation: layers. Parameterized

Resource<C, S, P, O> : CSPO : URIs hierarchy.

Materialized interactions re-populate model and context (Augmentation). Browse context model: kinds and grammar known statement "templates" (by kinds hierarchy layers aggregation) navigation for discovery of domain messages resource kinds.

Model, Context, Interaction IO: Message. Nested CSPO contexts quad, CSPO resources (plain URIs, kinds, nested contexts). Wildcards, variables, placeholder, null values: Message structure defines CRUD behavior.

Message: Resource model hierarchy parent class (monad of plain URI, parameterized resources). Resource set specification. Any Resource is a Message, specifying a potential set of other Message (Resource) in a model (layers).

Resource : Message. Resource resolution: known URIs, known resource kinds bindings, dialog (resource set specification) recursively. Interaction model (dialog resolved resources set). Wildcards, variables, placeholder, null values: Message structure defines CRUD behavior.

Resource monad of URIs or Message monad of Resource?

///

**Encodings**

aX\*4 + bY\*3 + cZ\*2 = dW

d, a, b, c: classes (CSPO)

WXYZ: instances (CSPO)

powers: CSPO role

terms: CSPO resources

Z(obj) is Y(pred) for X(subj) in W(ctx)

instance, class, metaclass, ocurrence terms.

primitives, variables, placeholders.

resolution (Discovery, DIDs).

Templates (grammar)

Subjects: attr / val, ctx / role

Behavior: order / compare.

Proof of work

MetaGraph model: map URIs -> IDs

Satisfy dW. Sync resolution (recurse terms contexts)

FCA. Resource attributes. Tensor, adjacency matrix, tree.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

Dimensional / Grammar models.

///

**Reactive Dataflow. Resource / Message / Transform. Behavior in graphs.**

Message flow (event loop) in / out: Alignment (data) <-> Activation (schema) <-> Aggregation (behavior)

Encode behavior in statements / graph:

Comparisons, order. Sort. Order (kinds hierarchy?)

Pattern matching, iteration, jumps. Discovery: routes / signatures, next event in bus / graph.

Context Model Message: Resource Specification (Grammar Template). Messages Model: context model instance from input model grammar. Transform: context model instance from Messages.

Express Augmentation (Alignment, Activation, Aggregation) as Messages / Transforms. Reified Model entity types / roles (CSPO, Kinds, Layers, etc.).

Resource monad of URIs or Message monad of Resource?

Encoding. Addressing. Schema / MetaModel for data (Model), schema (Context), behavior (Interaction) resources / layers (aggregation). Naming formats / schemes: namespaces, contexts. Class hierarchies (express context / class / kinds hierarchy). Dimensional metadata. Resource MetaGraph bindings (Message expansion / resolution index).

Subscription, Subscriber, Producer, Consumer, Processor.

Example: submitting Behavior layer grammar / context "template" initiates "dialog" for fulfill Behavior expanding Message(s) and nested context layer statements (known / resolvable, new behavior / subitems) needed to complete / update full Behavior layers contexts graph.

Augment. Alignment, Activation, Aggregation Message(s) : Resource set specifications.

Model listens onMessage (interaction context model population / dialogs scopes / namespaces).

Model augments input Message (augmentation specifications over in Message).

Model expands Message (Message over model resources):

Resource listen modelMessage. Model subscribes to response.

Matching triggered Resource. Message matching semantics (transforms).

Triggered Resource publish itself modelMessage.

Model augments output Message (augmentation specifications over out Message).

Model publish onMessage (interaction context model dialogs / resource dumps).

///

**DCI Metamodel (Base Model)**

(Resource, Resource, Resource, Resource) : Resource (Model).

(Statement, Subject, Attribute, Value);

(Role, Statement, Attribute, Value);

(Kind, Role, Statement, Attribute);

(Class, Kind, Role, Statement);

(Flow, Class, Role, Entity);

(Behavior, Flow, Class, Role);

Messages (Model : Resource) as Resource set specifications. Subject, Attribute, Value : Resource.

Determine class (reified layers contexts) hierarchies:

(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model resources, statements, kinds.

Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

Models definition: data (Statement, Entity), schema / contex (Role, Class), interactions / behavior (Flow, Behavior).

Kinds / Roles:

Grammar: kinds layers aggregation (CSPO layers Kinds).

Layers: Roles (Models metaclass context resources).

Reified Kind: (Kind, Occurrence, Attribute, Value);

Grammar input set model specificatíon (Statement layer kinds).

Dimensional input set model specificatíon (from Statement layer, ordered SPOs: order criteria, comparisons. Kinds / class / occurrence / instance order criteria?). Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements layers).

///

**Dimensional Metamodel**

(Value, Distance, Previous, Next);

(Measure, Value, Distance, Previous);

(Unit, Measure, Value, Distance);

(Dimension, Unit, Measure, Value);

(Concept, Dimension, Unit, Measure);

(Resource, Concept, Dimension, Unit);

(Statement, Resource, Concept, Dimension);

Populate / align / annotate models with dimensional data. Model input: statements (model resources). Model specification: augment, sort statements. Model specification: specialization of base model layers. Resolve resolution statements order.

Dimensional input set model specificatíon (from Statement layer, ordered SPOs: order criteria, comparisons. Kinds / class / occurrence / instance order criteria?). Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements layers).

Value -> distance(prev, next); ordering;

Assert knowledge: 1h -> 60min, lun-mar-mie-jue-vie, 1mt -> 100cm.

Comparison / order: Alignments (prevv, curr, next). Next hour, location, city, country, next distance at next time at current speed. Event sourcing / tracking: married -> marriage occurred.

Sort: cause / effect, temporal, etc. Messages align, functional map, fold, etc. Primitives.

Encode layered statements ordering. Complement / suplement concepts definitions.

Events metamodel (TBD):

(Object, State, Axis, Type)

(State, Axis, Type, Event)

(Axis, Type, Event, Event)

(Type, Event, Event, Event)

(Event, Event, Event, Event)

Model MetaGraph (TBD):

(ResourceClass, ResourceID, Statement, Kind);

(StatementClass, StatementID, ResourceID, Kind);

(KindClass, Kind, StatementID, Kind);

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

Ontology matching (table, pk, col, val example). Helper upper models for models linking / alignment.

Events declarative definition. State change of value in axis in measure of context.

Messaging metamodel:

(Message, Resource, LHS, RHS);

(Interaction, Message, Resource, LHS);

(Role, Interaction, Message, Resource);

(Context, Role, Interaction, Message);

(Dataflow, Context, Role, Interaction);

///

**Event sourcing (“offline” sync). API**

Graph linking / alignment / sinchronization by entailments from event sourcing over inferred state.

DOM / OGM APIs (JAF).

I/O Implementation, Deployment.

Model, URI, Resource, Statement, Kind hierarchies. Models architecture (URI class per layer).

DIDs / P2P / Rx Implementations.

Model API. ModelManager.

Event loop. IO.

Protocol: Input statements for querying augmented knowledge. Browse result model graphs. Input statements encoding queries / commands: grammars, reified message contexts (templates / forms). Browseable models, contexts, interactions (state / content semantic activation). Dataflow according Messages input signatures.

**Application**

First, I'll try to describe a "problem" (problem "spaces" in this case) and how a Purpose driven user Community achieves its Goal(s) by means of Goods, Products and Needs satisfaction (ontology levels: from abstract upper ontology to user gesture command in user interface / service invocation).  
  
The problem is to organize interdisciplinary (multiple domains) Task(s) in a Purpose fulfilment network with Actors, Contexts and Roles (with attributes and values). Problem spaces (domains) are declaratively stated by DCI[1] design pattern: Data / Context / Interaction use cases definitions and instances.  
  
Collaborative Federated Actor network complying determinate Profile(s) satisfying specific Product / Good / Need abstraction playing determinate Role in use cases Context.  
  
Domain Translation between business domains, example: orders, delivery, invoicing (micro) services Model instances are the means by which distributed disparate data, schema and behavior of different sources (applications, services) integration could be performed by means of Semantic Intelligence and Augmentation Protocol(s).  
  
A domain can be defined in terms of a set of actions / tasks with the Purpose of satisfying some Goal solving the Need for a Good producing / gathering a Product. Ontology. Purpose as Goal “class”.

The principal focus is to deploy a (social) Collaborative peer (Actor) network for which entities and individuals develop Profile(s) which acquaint them with Purpose resolution capabilities. Then, according peer’s specific needs (domain Goals) the application orchestrates interactions needed for Product(s) Task(s) accomplishment.

URIs API for annotating network retrieveable resources metadata. Content type / model driven augmentations / activations (models features / outputs). Subject attributes / values. Occurrences contexts / roles. Paths, pointers, locators. Example: annotate document URIs (parts, sections, mentions), annotate images URI (whole image description, coords: classes, individuals), annotate DB, table, row, column, value URIs, annotate / describe service / APIs URIs. Hypermedia protocol composable with other (described / annotated) APIs / resources. Example: Drive APIs.

What my attempts are about where, in the beginning, to match different URIs or 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.  
  
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);  
  
Each layer abstract:  
  
Statement (data instance):   
(Statement, Occurrence, Attribute, Value);  
someOne buys someProduct  
  
Entity (data class):  
(Entity, Statement, Occurrence, Attribute);  
someBuyer, someProduct (Entity);  
  
Role (schema instance):  
(Role, Entity, Statement, Occurrence);  
Buyer, Product (Role);  
  
Class (schema class):  
(Class, Role, Entity, Statement);  
Person, Good (Class);  
  
Flow (behavior instance):  
(Flow, Class, Role, Entity);  
someBought (Flow);  
  
Behavior (behavior class):  
(Behavior, Flow, Class, Role);  
Buy (Behavior);  
  
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 metamodel. The act of applying an Augmentation implies one source Resource (context), one template Resource (transform) and a resulting (set of) Resource(s).  
  
One also could Augment Resource(s) in a functional manner, using reactive event driven APIs so, for example applying "Person" class to "Employee" role could shield a Resource set of people being working for someone. The ultimate goal is to be able to "plug" as much "backends" connectors as posible into distributed peers which exposes protocols / APIs for knowledge driven hypermedia applications.

Application:

features / techniques / patterns.

Implementation deployment use cases. Sample Apps: SoLiD / PIM / PASCEN: App declaratively built with framework, Implementation Integrations.

Extension / Augmentation: BI / EAI. Smart dashboards / reports / workflow / process / activity components. Activable smart indicators / components (predict / execute). Declarative Model interpretation into abstract application models. Rendering (Gestures ontology).

**RDF / OWL, Graphs, Triples, Quads introduction.**

Serialization. TBD.

**Model: Object Graph Representation as RDF Quads.**

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).

**URIs, Resource, Statement, Layer, Kind APIs.**

Context / Resource type hierarchy design pattern: plain class hierarchy, parameterized class on Resource(s) / URIs, monads, metaclass, others. Actor / context / role (Statement CSPO position / Meta Resource). Reified Model types. DOM.

Meta Resource(s): URI, Resource, Statement, CSPO, Context / Layer, Occurrence, Attribute, Value, Kind, etc.

DOM, Actor / Role / Context, OGM APIs.

Augmentation: transform algorithm (basic operation).

Encoding: Model (Resource).

Model: RDF Backend.

URIs Services: API for plugging whatever connector may be implemented for behaving in a reactive message oriented fashion (back ends).

Resource: Abstracts (wraps) URIs Services in a functional API (Resource streams). DOM, Actor / Context / Role (Meta Resources).

Augmentation: Parse Message (event: context quad) according Template (pattern), materialize output Transform. Algorithm (TBD): case classes, pattern matching, destructuring, Resource monad chained operations (Template: functor) functional streams, ADTs.

Dataflow, Reactive: Resource Monad handling of wrapped URIs messages / events I/O via HTTP verbs. Augmentation: Model, Context instance / class (layers), Resources producing / reacting to events. Endpoints: Discovery / Location / Resolution services. URI APIs (signatures discovery).

Meta Graph / Model, Meta Resource(s): Resources / Messages reifying "patterns" on inputs (URI, Resource, Statement, Kind(s), Context, Occurrence, Attribute, Value, Layer Context classes, etc.). Declarative statement for Augmentation shapes applyied to input contexts.

Meta Model default Augmentations:

Aggregation classification. Registry svc.

Alignment regression. Index svc.

Activation clustering. Naming svc.

Context Kind Signatures.

Datasources / Backends / Services. URIs. Signatures: dataflow (Context Kinds). CKs Attribute / Value (SK / PK) determines domain / range I/O of a Resource / URIs.

Ontology matching (Backend / Interaction Model).

Model Meta Resource: Model components reified Resource types / instances (URIs, Resource, Statement, Context : Layer, Kind, etc.). Augmentation templates "placeholders" (signatures, matching of common upper resources).

Kinds (Application):

Kind: Basic type inference. Applied over layers CSPO during Activation Augmentation. An Occurrence Attributes / Values, aggregated for its URI and Context, determines Kind "members" (Attribute) and Kind instance member values (Value).

Super Kind / sub Kind hierarchy relationship is given by a set of Kind Attributes being super set / sub set of each other.

Examples.

SubjectKind (meta Resource): For a given URI occurring as Subject (Occurrence) across a set of Statements (Contexts), its aggregated Predicates (Attributes) defines its "Kind" and its Attribute values determines the given Kind instance "members" values.

ObjectKind (meta Resource): for a given URI occurring as Object (Value) over a set of Statements, Subject (Kind Attribute), Predicate (Kind Value).

PredicateKind (meta Resource): for a given URI occurring as Predicate over a set of Statements, Object (Kind Attribute), Subject (Kind Object).

ContextKind: SubjectKind (Attribute), ObjectKind (Value). Context (Statement) "signature" (dataflow inputs / outputs activation: domain / range).

**Functional Implementation: URI / Resource APIs.**

Model state: Context (Resource : data), Kind (Grammar : schema), Dimension (behavior). Context Kind(s) signatures: Dataflow.

Augmentation: basic operation.

Monad: Resource<URI>.

Resource layers hierarchy API.

Data / Reference Model. Model Functional Semantics (Model / Layer / Message application). Augmentation: Basic Model I/O operation. Message spec / Resource Set Specification (result).

Service URIs:

Service URIs: Context Kind (inputs / outputs domain / range). Example: predictions, classification, clustering, regression. Index / Naming / Registry "contexts" (facets).

Extended content types activations on domain / range (verbs, augmentations). Example: image, face, crop.

Functional Resource Model / Context / Attributes / Kind design / implementation. Serialization (Encoding / Models). Signatures. Reactive. Augmentation. DOM, Actor / Context / Role. APIs: Augmentation. Meta Resources.

Meta Model: Encode / reify Model(s) declaratively w./ Meta Resources and Model Context(s) hierarchies.

Meta Model: Encode Kind / Context hierarchies.

Meta Model: Encode order, iteration, conditional flow. Dataflow.

Functional Resource Model / Context / Attributes / Kind design / implementation. Serialization (Encoding / Models). Signatures. Reactive. Augmentation. DOM, Actor / Context / Role. APIs: Augmentation.

Resources API hierarchy.

Meta Resources.

Meta Model: Encode / reify Model(s) w./ Meta Resources and Model Context(s) hierarchies.

Meta Model: Encode Kind / Context hierarchies.

Meta Model: Encode order, iteration, conditional flow. Dataflow.

Augmentation / Models: Source, Grammar, Dimensional Models. Core Meta Model Augmentation Template(s): Encoding signatures Dataflow.

Functional Resource Model / Context / Attributes / Kind design / implementation. Serialization (Encoding / Models). Signatures. Reactive. Augmentation. DOM, Actor / Context / Role.

Meta Resources.

Meta Model: Encode / reify Model(s) w./ Meta Resources and Model Context(s) hierarchies.

Meta Model: Encode Context hierarchies.

Meta Model: Encode order, iteration, conditional flow. Dataflow.

Encoding: Kind hierarchies / Grammars (CK, SK, PK, OK).

Encoding / Models: Source, Dimensional Models. Encoded Grammar Template(s).

Augmentation: declaration (signatures) / algorithm.

Ontology Matching. Semiotic. Sets. Functional Reference Model.

**Services (URIs APIs)**

Index

Naming

Registry

Service (URIs APIs). Index. Naming. Registry. Custom (signatures : Context Kind).

**Data / Reference Model.**

Functional declarative Semantics Specification. Semiotic / Dimensional alignment layers. TBD.

**Ontology matching. Ontology levels.**

Semiotic / Dimensional alignment. TBD.

Ontology Matching. Semiotic. Dimensional. Sets. Functional Reference Model.

Semiotic / Dimensional alignment / aggregation layers (lower resource alignment layers):

(Context, Sign, Concept, Object);

(Value, Distance, Prev, Next : in Units); (Measure, Value...) (Unit, Measure, Value,...); (Resource, Unit, Measure, Value); Marriage example.

Messaging metamodel:

(Message, Resource, LHS, RHS);  
(Interaction, Message, Resource, LHS);  
(Role, Interaction, Message, Resource);  
(Context, Role, Interaction, Message);  
(Dataflow, Context, Role, Interaction);

Meta Model (Meta Resources)

Semiotic / Dimensional (encode matching Resources). Common upper ontology matching layers. Models:

Source Model. Data.

Grammar Model. Schema.

Interaction Model: Behavior?

Ontology matching (table, pk, col, val example). Helper upper models for models linking / alignment.

Meta Resource(s): URI, Resource, Statement, Model, CSPO, Layer, Context, Occurrence, Attribute, Value, Kind, etc.

Semiotic encoding:

(Context, Sign, Concept, Object);

Object as Sign: Concept: Attribute. Other mappings (roles).

Semiotic / Dimensional Alignment, Aggregation (known mappings) : Class / ID Ontology Matching. Contextual IDs (infer occurrence contexts). Inference ID lookup of ID for desired satisfaction of given transforms / roles / operations.

**Model Layers:**

Augmentation: basic operation.

Layered data, schema, behavior class / instance quads hierarchy. Model layers: URI quads:

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);

Graph Execution Semantics: Dataflow by Context Kind domain (Subject Kind) / range (Object Kind).

Ontology Matching. Upper ontologies. Primitives.

**Addressing / IDs / Encoding.**

Encoding: Resource ID. Encoded Resource contents (signature / occurrence). Augmentation: Resource set (Message) resolution from context over Template / Resource(s).

Encode IDs: Context Kind, upper (meta) Resources (levels / layers). Resource contents / contexts (identify by occurrences in roles in other contexts, Meta Resources, layers class, metaclass, instance).

Encode common upper Semiotic / Dimensional Model: Reference Model.

Encode Kind / Context hietarchies.

Encode Augmentation(s) as Resource descriptions.

Encode Model(s) as Respurce set. Meta Resources, layers Contexts, Kinds (reified).

Encode Graph Execution Semantics. Dataflow: Context Kind signatures. Iteration, conditional jumps.

Events / Messaging.

URIs, metaclass, class, instance, context, occurrence IDs. Formulae.

Resources wraps URIs streams sources / sinks activated by ontology matching alignment. Aggregates same entity different URIs, representations in contexts.

Context Kind / Signature: Predicate Kind from Subject / Object Kind.

Object occurrence of Predicate.

Encode behavior: iteration / jumps. Order statements (URIs APIs).

Meta Resource(s): Resources / Messages reifying "patterns" on inputs (URI, Resource, Statement, Kind(s), Context, Occurrence, Attribute, Value, Layer Context classes, etc.). Declarative statement for Augmentation shapes applyied to input contexts.

Sets. Quads.

Metaclass / Class / Instance.

Class / Instance ID pairs:

Subject / Context / Role : Attribute, Value. Metamodel. Encoding: each type as each (pair) kind. Pairs.

Semiotic encoding:

(Context, Sign, Concept, Object);

Value as Occurrence of Attribute in Attribute Occurrence Context. Meta Resource context roles).

Augmentation. Transform. Backend. DIDs: events sourcing (decentralized persistence). Encoding: avoid / resolve duplicate transactions.

Encoding: Resource ID. Encoded Resource contents (signature / occurrence). Augmentation: Resource set (Message) resolution from context over Template / Resource(s).

Augmentation: Message signature matches Template signature (across types hierarchies): Transform results Resource(s) for Augmentation predicates / mappings. Mappings: Meta Resources, Patterns,  Augmentations (in contexts), common hierarchy super Resource. Variables, expressions

**Message:**

Augmentation: basic operation.

Resource Set Specification (Statement) matching Model which returns augmented Message response (Model I/O).

Augmentation declarative Model definitions.

Message Resolution Algorithm.

Protocol: Augmentation Message dialog I/O.

**Models:**

Meta Model: Model Source, Grammar, Interaction facets specification.

Meta Model facets inputs aggregating Context(s) from layers. Upper alignment and augmentations. Reified.

Source facet input: Model Statement(s). Data.

Grammar facet input: Kind(s). Schema.

Interaction facet input: Flow(s). Behavior.

Meta Model: Model Source, Grammar, Interaction specification.

Source input: Statement(s). Data.

Grammar input: Kind(s). Schema.

Interaction input: Flow(s). Behavior.

Models: Meta Model / Resources. Model source / grammars / interactions. Upper semiotic / dimensional layers.

Layers / Contexts: Meta Model. Semiotic, Dimensional (upper). Source. Grammar, Interaction.

Models hierarchies aligned with Interaction Model. Source, Metagraph, Dimensional, Grammar.

Serialization. Encoding. Dataflow. Augmentation.

Explain layers, Meta Resource(s), Context (class / instance / metaclass) / Kind hierarchies. Augmentation behaviors description.

Model Contexts: Meta Model Meta Resources reified Contexts hierarchies. Models:

(Model, Behavior, Flow, Class); Model aggregation layer.

Meta Model (Meta Resources)

Semiotic / Dimensional (encode matching Resources). Common upper ontology matching layers. Models:

Source Model. Data.

Grammar Model. Schema.

Interaction Model: Behavior?

Metagraph Resource(s): class / instance IDs of reified meta Resource(s) in contexts / roles with attributes / values. Describes Model(s) : Interaction Model (Source, Dimensional, Grammar).

Resource: reactive entity. Augmentation: apply Interaction Model / input Message to parsed Resource. Reaction: matching Resource set (resolution depending Resource type).

Message: Resource aggregation (occurrence, context, model) dataflow (Augmentation). Resolves Resource Set specification.

From Intetaction Model Augmentation (patterns: CRUD / IO, Aggregation, Alignment, Activation): Source, Grammar, Metagraph, Dimensional models. TBD: Parser (consumes Resource inputs, apply Message rules, emits Resource set).

Grammar (kinds), Metagraph (contexts, meta Resource roles): Contextual / Functional Type Object (Dynamic Object Model), Actor / Role pattern models.

Kind in context: URI / Resource<T extends URI> Monad (Type Object).

Role in context: URI / Resource<T extends URI> Monad (Actor / Role).

Context: CSPO Occurrence. Actor role meta Resource.

Types / Roles: Reified Kinds / meta Resource(s).

Model Contexts: Meta Model Meta Resources reified Contexts hierarchies. Models:

Data: Source / Interaction, Schema: Encoding / Grammar, Behavior: Dimensional / Measures.

(Model, Behavior, Flow, Class); Model aggregation layer.

Ontology Matching. Semiotic. Sets. Functional Reference Model.

(Context, Sign, Concept, Object);

Dimensional alignment / aggregation layers (lower resource alignment layers):

(Value, Distance, Prev, Next : in Units); (Measure, Value...) (Unit, Measure, Value,...); (Resource, Unit, Measure, Value); Marriage event example.

Model Contexts: Meta Resources / Contexts hierarchies. Models:

Data: Source / Interaction, Schema: Encoding / Grammar, Behavior: Dimensional / Measures (marriage).

(Model, Behavior, Flow, Class); Model aggregation layer.

Ontology Matching. Semiotic. Sets. Functional Reference Model.

(Context, Sign, Concept, Object);

Semiotic / Dimensional alignment / aggregation layers (lower resource alignment layers):

**Interaction (Meta) Model Specification.**

Aggregation (data)

Alignment (schema)

Activation (behavior).

Align to: URIs, Resource, Statement, Kind, Context Kind, Context, Occurrence, Attribute, Value.

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

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);

Statement Aggregation: Statement instance Context for each distinct CSPO URI on inputs aggregates same URI Occurrence as Subject with corresponding Attribute (output Predicate) / Value (output Object). According CSPO input as Occurrence, corresponding Attributes / Values are chosen.

Resource : Functional URI wrapper.

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

(Statement, Occurrence, Attribute, Value);

Data: Aggregation layer: for each previous layer Message, layers: (Aggregation Instance, previous Message Context as Subject, previous Message S/P as Attribute / Value). Previous layer: Aggregation until end of source Messages layers (6 Aggregation statements consuming previous CSPOs. Renders to Aggregation instance contexts of Aggregation class).

Schema Alignment layer: Context / Occurrence / Attribute / Value. Renders augmented Attribute / Value Context / Occurrence.

Behavior: Activation layer: for each layer Message, Activation (Kind instances) are for each Activation class taking one of Message CSPO as Kind Subject and their corresponding CSPOs as Attribute / Value. Kind classes for each Aggregation layer. Context Kind: composite Subject / Predicate Kinds as Attribute / Value.

Layers dataflow: hierarchical Message inputs / outputs.

**Source Model Specification.**

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);

**Metagraph Model Specification.**

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);

Metagraph / Grammar (sample):

(Kind, SuperKind, Attribute, Value);

(Occurrence, Kind, SuperKind, Attribute);

(Context, Occurrence, Kind, SuperKind); (attributes / links bindings).

(Resource, Context, Occurrence, Kind); State Resource Kind in occurrence context (context / role bindings).

(Statement, Resource, Context, Occurrence); State Resource URIs occurrences / Resource class IDs (classification bindings).

(Interaction, Statement, Resource, Context);

(Action, Interaction, Statement, Resource);

Interaction / Model?

Action / Schema?

**Dimensional Model Specification.**

(Value, Previous, Distance, Next);  
(Measure, Value, Previous, Distance);  
(Unit, Measure, Value, Previous);  
(Dimension, Unit, Measure, Value);  
(Concept, Dimension, Unit, Measure);  
(Resource, Concept, Dimension, Unit);  
(Statement, Resource, Concept, Dimension);

Example:

(Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.

Order layers statements. Hierarchies (contexts / kinds). Parent / child relationships (steps). Order type relationships: husband: single / marriage / married.

(Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.  
(Measure, Value, Previous, Distance);  
(Unit, Measure, Value, Previous);  
(Dimension, Unit, Measure, Value);  
(Concept, Dimension, Unit, Measure);  
(Resource, Concept, Dimension, Unit);  
(Statement, Resource, Concept, Dimension);

Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements layers).

Value -> distance(prev, next); ordering;

Assert knowledge: 1h -> 60min;

dom-lun-mar-mie-jue-vie-sab (orders);

1mt -> 100cm;

etc.

Comparison / order: Alignments (prev, curr, next asserted knowledge). Next hour, location, city, country, next distance at next time at current speed. Event sourcing / tracking: married -> marriage occurred.

Sort: cause / effect, temporal, etc. Messages align, functional map, fold, etc. Primitives. Encode layered statements ordering. Complement / supplement concepts definitions.

Events metamodel (TBD):

(Object, State, Axis, Type)  
(State, Axis, Type, Event)  
(Axis, Type, Event, Event)  
(Type, Event, Event, Event)  
(Event, Event, Event, Event)

**Grammar Model Specification.**

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);

**Interaction Model:**

Augmentation: basic operation.

Source (upper) Model. Models hierarchies aligned with Interaction Model.

Interaction Model provides event sourcing, distributed inference / synchronization (distributed consolidation and alignments).

Interaction Model I/O : Message (from URIs or events) perform and materialize applying Augmentation from Interaction Model population.

Message declaratively states Model Specification through Message Augmentations.

Meta Resource(s): Resources / Messages reifying "patterns" on inputs (URI, Resource, Statement, Kind(s), Context, Occurrence, Attribute, Value, Layer Context classes, etc.). Declarative statement for Augmentation shapes applyied to input contexts.

Augmentations (core Meta Model):

Data (Aggregation);

Schema (Alignment);

Behavior (Activation);

Interaction (Meta) Model Specification (Metacircular interpreter: encodes Model(s), including itself): Interaction Model reifies / declaratively renders Source, Metagraph, Dimensional, Grammar Models via Augmentation Specification Message(s) from which it is populated and to which Augmentation (input Message) is performed, populating corresponding Model Resource(s).

Functional (monadic) Message Resolution Algorithm. Encoding.

**Augmentation:**

Augmentation: basic operation.

Augmentation: metamodel / custom (domain).

Message - Model - Template - Augmentation - Transform - Model - Message.

Encodings: Models

Functional / Signature IDs.

Grammars.

Message: Resource.

Model event. Data.

Resource ID / Set specification.

Model: RDF. Resource layers.

Reified Models. Upper (Semiotic / Dimensional) layers Alignment. Ontology Matching.

Template: Resource. Grammar.

Model state. Context.

Functors.

Augmentation:

Model I/O, Dialog. Interaction.

Algorithm: parsing, declarative.

Transform: Resource.

Results (dataflows).

Materialize.

Model

Message

Ontology / Persistence.

Functional Reference Model.

CRUD (events).

Augmentation: Basic Model I/O operation. Apply Model / Service (layers dataflow) to input Message quads. Layer. Dialog.

Messages Resource Set Specifications for CRUD, Aggregation, Alignment, Activation over Model. (Interaction Model Specification) stated on Interaction Model or from Protocol Message.

Model I/O: Augmentation Message application over Model from backend (URIs) Message or from Model I/O (layers) Message. Returns Resource Set populated / materialized Message.

Model I/O: layers application. Output model layers classes (layer Context) as stated in Interaction Model for input Message.

Model I/O: application of layer context class, state context, occurrence, attribute, etc. placeholders (value of placeholer in inputs) via reified statement roles in CSPO of layer statement specification (output).

Augmentation state Occurrence aggregation of Attribute / Values (i.e.: Statement / Roles), CSPO rendering / translation to output Message and transforms as specified in Intetaction Model.

Augmentation: each Augmentation populates corresponding Models performing CRUD, aggregation, inference and classification augmentations from Interaction Model Specification.

Layers. Augmentation: new IDs / ID Contexts. Naming.

Resolve Message matching Resource from behavior layers / matching kinds from Model / data layers.

(Kind, SuperKind, Attribute, Value);

(Occurrence, Kind, SuperKind, Attribute);

(Context, Occurrence, Kind, SuperKind); (attributes / links bindings).

(Resource, Context, Occurrence, Kind); State Resource Kind in occurrence context (context / role bindings).

(Statement, Resource, Context, Occurrence); State Resource URIs occurrences / Resource class IDs (classification bindings).

(Interaction, Statement, Resource, Context);

(Action, Interaction, Statement, Resource);

Example: a message composed of a kinds CSPO matches statements “instances” of those specifications (statements whose CSPO have matching kinds). A message with three CSP kinds and a (potentially unknown) object URI retrieves matching resources having that object value into corresponding property kinds. An statement of plain (potentially unknown) URIs instantiates / updates and augments new / known resources added to models and returns an augmentation transform result.

Interaction Model: Context of Messages model for a given interactions session / dialog state. Message invocation requests: Statement(s) building Resource invocation graph with layers matching Message patterns. Layers graph invocation patterns matching from higher to lower layers resources fulfilling higher layers templates. Variables, wildcards, placeholders.

Dialog arguments resolutions example: higher layer Resource / Message request / invocation instantiates in Interaction Transform context corresponding lower layer graph statements to be “populated” to fulfill request. Message IO of “forms” (Messages) inter-peers (originating peer  
acting as “server”) for initial requested peer to “ask” for form elements to be populated (interaction context “dialogs”). Resolution may propagate to other peers (content aware addressing dataflow routes dispatch: P2P resources address encodings, matching forms models requests). Nested interactions.

Explain messages (resource resolution). Grammar. Match model Resource(s). Compound nested CSPO statement contexts defines result behaviors. Message CSPO contexts may define create, retrieve, update or delete operations (passing 'null' for example for resource / statement to be deleted).

Explain transforms (message application). Transform: Resource stream result of Message application over resolved Resource(s)). Input statements: Message(s) / Resource(s) (from input message or to be populated or populated in dialog) and "goal" Message / Resource aggregating a model from Resource MetaGraph with Message / Resource bindings.

API: URI, Resource, Message, Statement, Kind, Layers. Representation: XML bindings.

Kind : Statement : Message : Resource : URI;

URI / Resource<T extends URI> : Monad.

Resource: (URI, URI, URI, URI); URI : Resource.

Message: specification / transform (input / output dialog domain / range). Context Kind.

Augmentation / Models: Source, Grammar, Dimensional Models. Core Meta Model Augmentation Template(s): Encoding signatures Dataflow.

Encoding: Resource ID. Encoded Resource contents (signature / occurrence). Augmentation: Resource set (Message) resolution from context over Template / Resource(s).

Augmentation: Message signature matches Template signature (across types hierarchies): Transform results Resource(s) for Augmentation predicates / mappings. Mappings: Meta Resources, Patterns,  Augmentations (in contexts), common hierarchy super Resource.

**CRUD (I/O Message) Augmentation:**

Augmentation: CRUD (I/O Message).

Specification Model: Source.

Augmented Models (materialize, aggregate, align, activate).

**Aggregation Augmentation:**

Augmentation: Context Aggregation. Specification Model: Metagraph. Classification (aggregate quads contexts context / roles / class / identity).

**Alignment Augmentation:**

Augmentation: Data Alignment. Specification Model: Dimensional. Clustering (inference of links / attributes).

**Activation Augmentation:**

Augmentation: Interaction Activation. Specification Model: Grammar. Regression (classify roles in contexts: Kind).

**Model I/O Dataflow:**

Dataflow: Events. Reactive APIs.

Augmentation: basic operation.

Events declarative definition. State change of value in axis in measure of context.

Events: Dataflow. Reactive Model endpoint Message dispatch / resolution (Producer). Resolve (addressable) Message resources (Resolution template). Apply templates (Resolved resources : model / Message resources : view context) : XML (Message).

Layers (declaratively stated in Interaction Model):

Data input statements (Message).

Aggregate layers.

Align attributes.

Activate Kind.

Model: Reactive entity applying Message Augmentation resolving Resource Set Specification Message from inputs. Data Message (URIs layer), dataflow Message (Model / dialog).

Message Resolution Algorithm.

Data instance inputs (URIs events).

Model Message Augmentation resolution.

Interaction Model events / distributed / inference sourcing. Augmentations / CRUD: Interaction Model DIDs. URIs quad store / backend.

Augmentation. Transform. Backend. DIDs: events sourcing (decentralized persistence). Encoding: avoid / resolve duplicate transactions.

Resource: Reactive entity (events source / sink) wrapping an URI endpoint implementing some kind of I/O, Signature: Resource Context Kind. Matching “ranges” (SK) dispatch matching events to matching “domains”.

DIDs: Encoding (signature / contents) identifier. Endpoints: provenance. Address: Messaging bus. Discover signatures, contents, potential transform results.

Dataflow:

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

Addressing. Reactive (Events, Dataflow). Graph encoded behavior (encoding / patterns). Reactive objects (Model, Layer / Statement, Resource, URI). Dispatch: Bus / DIDs resolution.

Augmentation. Transform. Backend. DIDs: events sourcing (decentralized persistence). Encoding: avoid / resolve duplicate transactions.

Model

Message

Interaction

Transform (Augmentation)

Flows / Routes (Augmentation, signatures)

Addressing

IDs Encoding

Processor

Producer

Consumer

Subscriptions (from metadata)

Queues.

**Protocols (Deployment / use cases):**

Dataflow, Reactive: Resource Monad handling of wrapped URIs messages / events I/O via HTTP verbs. Augmentation: Model, Context instance / class (layers), Resources producing / reacting to events. Endpoints: Discovery / Location / Resolution services.

Augmentation: Model, Context (Statement), Resource levels Message (quads) IO application, resolution, transform / declarative specification (template, input context, results). Dataflow contexts from Message levels application.

Augmentation: For example, a template Statement (Statement used as transform specification) from, for example, the Interaction Model, may state matching pattetns such as:

(ContextClass : Subject, Context, Occurrence, Attribute);

and, when applied to an input Message:

(Statement, Subject, Predicate, Value);

reacts emitting the following Statement, transforming input context Message according template rules (input Subject -> output Attribute):

(TransformClass : Entity, Statement, Subject, Predicate);

which is materialized in the corresponding Model and is itself again a Message routed for further processing. TransformClass is an instance / subclass of super / meta class ContextClass (model layers transform rules).

Augmentation contexts / templates: Model, Layer, Resource. Template Meta Resource(s) (Context, Occurrence, Attribute, Value, CSPO, Kind, etc.): matches context input Message Resource by context extending / implementing / instantiating such Meta Resource(s).

Transforms: explicit template resources / model layer resources as input / specification (i.e.: apply a Role to a Class from Source Model: Entities playing such Role as results). Model Resource as template outputs common supertypes with context input as Message result.

Augmentation. Dialog. Query API.

Forms. Templates.

Ontology levels / layers.

Augment / Activate Resource (via addressing).

Extension / Augmentation: BI / EAI. Smart dashboards / reports / workflow / process / activity components. Activable smart indicators / components (predict / execute). Declarative Model interpretation into abstract application models. Rendering (Gestures ontology).

**Protocols (Deployment / use cases):**

Hypermedia addressing and annotations. Extended content types annotations: request accept: image/png;people, response content type: text/xml;facesCoords.

Addressing: according content type (i.e.: response XML dialect for coordinates in an image / hash determining anchor in an HTML document) renders corresponding object (DOM document in this case) for “activation” on addressed parts.

Context signatures. Signatures activation (JAF) interactive dashboards.

Activation (parse gestures / render content according context). Browser.

URIs scheme. Extended Content type. Message dialog (peers Augmentation).

Goal, Purpose: Fulfill Context.

Forms / Templates.

Dialogs: Model I/O (Message) flows.

Models browsing / discovery APIs.

HAL / OData like.

Platform:

Implementation (Protocols). Core, RX, Dataflow. Model: Reactive Dataflow.

(Resource : URI) : DID : Class / ID aligned Resource URIs.

DIDs encode Resource contents (hash / tensor / Context Kind) signatures. Resolution. Endpoints (provenance / contexts).

Resource: Reactive entity (Processor). DIDs: Resource Bus addresses. Container: services / nodes (models).

Bus / reactive dataflow layer (physical distributed Resource(s) events dispatch: services / nodes containers). Publish / consume Resource streams.

DID encoded Resource hash: events signatures.

Resource produced events (by Context).

Resource consumed events (by Context).

Encoding. Endpoints. Dataflow.

Augmentation: common super type inference: Aggregation, Alignment, Activation. Verbs / Activation. Functors (context: messages, reified mappings: templates).

Message: specification / transform (input / output dialog domain / range). Context Kind.

Augmentation: Aggregation (Context template).

Augmentation: Alignment (Attribute, Value template).

Augmentation: Activation (Kind type inference, Class / ID resolution / alignment: semiotic / encoding templates).

Augmentation templates: Metagraph.

Core Backend APIs.

Node Quad Store Backend. Sync DIDs.

RDF / OWL Backend URIs (Statement Context / Resource addresses, services).

DIDs: decentralized persistence. Event sourcing. Sync Backend. Identifiers for (reified) meta Resource (URI, Resource, Statement, Context, Kind).

Protocol / Dialog: I/O. Prompts.

Application Ontology Levels:

Backend

Session

Frontend / Service

Domain Ontology Levels (DCI layers). Application ontology Aligned.

Ontology levels: data / schema / behavior (backend, business, frontend) objects.

Application augmentations / extensions (connectors):

Microformat like frontend / services (rendering layer) elements annotations protocol (ontology levels / contexts vars: referer, data values: price, schema rels: master detail, behavior: account transfer) for hypermedia activation rendering layer. Annotations: addressable / addresses in rendering context.

Render Wiki like abstract representations for hypermedia rendering / activation.

XML abstract representation of reactive content / behavior declarative description. Extended content types. XLink, XPointer, XQuery.

JSON / XML / XSL: XUL / ZUL / HTML (rendering frontend / services layer formats). XSLT / XPath / XLink / XPointer / XQuery.

Resource XML Encoding (nested layers quads). Message XML Encoding.

XSLT templates (Resolution, Activation, Alignment, Aggregation). Resolution algorithm: TBD (ontology matching).

**Contents**

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.

Distributed P2P (Blockchain) approach of data synchronization between peers for ease of deployment patterns election and datasources integration (APIs, microservices, etc.).  
  
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, equivalent attributes, equivalent roles).  
  
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).  
  
Layered models semantic infrastructure for integration of heterogeneous backends (meta models).

Alignments Augmentations:  
  
Activation: type inference : classification (determine class / metaclass / roles for entity attributes and values).  
  
Activation infer attributes / relations : clustering (from multiple occurrences of same entity in diverse data sources).  
  
Aggregation: infer roles in contexts: regression (Person class in Employment interaction : Developer role).  
  
Integration of addressable resources. Reactive I/O (sync back ends). Content type driven semantic augmentation / annotations.  
  
Integrated view. Navigate contexts, data, interactions. APIs. Dimensional views annotations (analysis / mining).  
  
Augmentation of distributed resources. Annotations (Semantic / ML). API for resource / schema / interactions exploration / protocol for message based API "dialogs" execution. HAL (Hypertext Application Language), OData (REST) like interfaces.  
  
Example: Google Drive / Google Knowledge Graph APIs Augmented with ML / Semantic intelligence tailored for specific domains / application kinds.

**Features**

Graph encoding of data / schema / behavior. Dimensional / Grammar annotations. MetaGraph: augmentation / transforms (Messages). Features.

Parallel distributed graphs models augmentation / transforms synchronization (Messages). Event sourcing (distributed inferences). P2P / DIDs.

Augmentation. Ontology matching. Hypermedia augmentation protocol. Browser / Client APIs.

URIs API for annotating network retrievable resources metadata. Content type / model driven augmentations / activations (models features / outputs). Subject attributes / values. Occurrences contexts / roles. Paths, pointers, locators. Example: annotate document URIs (parts, sections, mentions), annotate images URI (whole image description, coords: classes, individuals), annotate DB, table, row, column, value URIs, annotate / describe service / APIs URIs. Hypermedia protocol composable with other (described / annotated) APIs / resources. Example: Drive APIs.

**RDF triples, quads introduction**

RDF Models: rdfs type, class, subClassOf, sameAs, reification when appropriate. RDFS. OWL (alignments).

RDF / OWL Backend: APIs. Details: Contents triples / models introductions.

Turtle. N3.

Example: feed Dimensional model for equivalences (units), comparison (orders).

TBD.

**Models: Quads, Contexts, Occurrences, Attributes, Values.**

Declarative means of using RDF quads to state application object models (data, schema and behavior).

Aggregation.  
Kinds.  
Grammar.

Formalization: Functional / Object API. Reference / Data model. Sets, categories, models.

Subjects: attributes / values, contexts / roles.

(Context, Occurrence, Attribute, Value);  
(Context, Sign, Concept, Object);

Instance, occurrence, class, metaclass.

Hierarchies: layered quad statements are represented by a class hierarchy which root is the Resource<T> monad. There is a subclass relationship between each layer implementing class and the one of the next layer (Dynamic Object Model).

Quads in the context role of lower layers represents occurrences of context enclosing layer. Assert class hierarchies, order relation (temporal, causal, containment, etc.) by attrs / vals, set / superset relations.  
Discovery: All model kinds are browseable / discoverable.

Determine class (reified layers contexts) hierarchies:  
(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model resources, statements, kinds. Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch,  
event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.  
Dimensional / Grammar models.

TBD.

**URIs, Resource, Statement, Kind APIs**

TBD.

Message service URIs: contextual (statement / dialog) service invocations.

Example: Subject (image URI / resource : source), Predicate (detection service / index service), Object (detection / search results endpoint / placeholder : destination).

Grammars: Predicate Kind (face / search recognition signature) from Subject (faces images / names) / Object (face classes / subjects) Kinds. Kind model layers.

Models definition: data (Statement, Entity), schema / context (Role, Class), interactions / behavior (Flow, Behavior).

Kinds / Roles:  
Grammar: kinds layers aggregation (CSPO layers Kinds).  
Layers: Roles (Models metaclass context resources).

Reified Kind: (Kind, Occurrence, Attribute, Value);  
Grammar input set model specificatíon (Statement layer kinds).

Dimensional input set model specificatíon (from Statement layer, ordered SPOs: order criteria, comparisons. Kinds / class / occurrence / instance  
order criteria?). Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements).

MetaGraph (resolution). Dimensional / Grammar alignments / annotations.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch,  
event bus routes. URIs / IDs mappings. Resource set specification resolution. MetaGraph resolves concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics via MetaGraph driven transforms (data / schema / behavior augmentation: dialogs).

URIs API for annotating network retrievable resources metadata. Content type / model driven augmentations / activations (models features / outputs). Subject attributes / values. Occurrences contexts / roles. Paths, pointers, locators. Example: annotate document URIs (parts, sections, mentions), annotate images URI (whole image description, coords: classes, individuals), annotate DB, table, row, column, value URIs, annotate / describe service / APIs URIs. Hypermedia protocol composable with other (described / annotated) APIs / resources. Example: Drive APIs.

Resource<T : URI> monad. Message functors. Transform reactive extensions.

Transform : Observer / Observable of Resource<T : URI>. Stream. Built upon Resources / Messages (TransformBuilder).

Identity and other core transforms (core messages). Stream.

flatMap(Message::apply) : Transform<Resource<R : URI>>.

API: Class for layer for model.  
API: Class for layer (DOM).  
API: Parameterized Resource: layer classes determined by URIs hierarchy, i.e.: Resource<Entity>, Entity : URI.

Base core services URIs (index, naming, registry). URI subclasses implementing / wrapping state for Resource monads offering protocols / addressing / content types / representations facades for services: DBs, WS (REST, SOAP, SPARQL), ML (predictions), etc.

Discovery: All model kinds are browseable / discoverable.

Determine class (reified layers contexts) hierarchies:  
(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model resources, statements, kinds.

Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

MetaGraph (resolution). Dimensional alignments / annotations.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

**Model Layers**

See Messages / Augmentation.

Composed of quads semantically aggregated into layers.

Core features provides:  
Alignment  
Activation  
Aggregation

Message / Transform driven specification of Alignment, Activation, Aggregation (Augmentation).

Message (Resource set expression);

Message: Model parent layer (Resource). Nested Messages CSPOs.

Model: Message scopes. Described as (nested) Message Resource set expressions.

Models: Data (Models), Contexts (Grammars), Interactions (MetaGraph : Models / Grammars bindings). Dimensional annotations.

MetaGraph: Resource, Statement, Kind class / instance as CSPO MetaGraph statement roles. Class / subclass relationship, Kind / subkind relationship.

Reactive nodes (Message events):

. Input Message event;

. Augmentation;

. Model / MetaGraph Message resolution (grammars / models / backends / services); Model Resource(s) response activation;

. Augmentation (Message : response / dialog);

. Output Message (events);

Grammars.

Upper / Dimensional ontology.

Inter models alignments.

Services (Endpoint URIs: Resource facades).

Reified model resources (CSPO, Resource, Statement, Kinds, Layers). Augmentation (Alignment, Activation, Aggregation) Messages / Transforms.

Model, Contexts, Interactions IO:  
Model: aggregated resource statements.

Context: aggregated model kinds (grammar statements).

Interaction: aggregated model / context bindings.

Inputs: resource statements, resolvable messages. Operation semantics (CRUD, browse, etc.) according input statements layout. Model endpoint. Materializes input resource statements and fully resolved message resource statements from interactions applying Augmentation and matching messages transforms.

Resource flow: input plain RDF URIs statements. Model / Context updates. Transform matches concrete resources.  
Resource flow: input message URIs statements. Context / Interaction perform. Transform matches resources in messages context grammar kinds hierarchies.

Outputs: resource statements with possible further resolvable messages (Model IO recursion / dialogs). Interaction queries context / model back for further resolutions. Message transform stream with request message applied plus matching context resolved resources from message.

Model, Contexts, Interactions IO:

Resource, Statement, Kind, Message, Transform.

Subscription, Subscriber, Producer, Consumer, Processor.

Model: aggregated resource statements model.

Context: aggregated model kinds (grammar statements model).

Interaction: aggregated model / context / dialogs bindings model.

Aggregation: layers. Parameterized Resource<C, S, P, O> : CSPO : URIs hierarchy.

Materialized interactions re-populate model and context (Augmentation). Browse context model: kinds and grammar known statement "templates" (by kinds hierarchy layers aggregation) navigation for discovery of domain messages resource kinds.

Model, Context, Interaction IO: Message. Nested CSPO contexts quad, CSPO resources (plain URIs, kinds, nested contexts). Wildcards, variables, placeholder, null values: Message structure defines CRUD behavior.

Message: Resource model hierarchy parent class (monad of plain URI, parameterized resources). Resource set specification. Any Resource is a Message, specifying a potential set of other Message (Resource) in a model (layers).

Resource : Message. Resource resolution: known URIs, known resource kinds bindings, dialog (resource set specification) recursively. Interaction model (dialog resolved resources set). Wildcards, variables, placeholder, null values: Message structure defines CRUD behavior.

Resource monad of URIs or Message monad of Resource?

**Data Model**

Data Model layers population / augmentation.

(Resource, Resource, Resource, Resource) : Resource / Message (Model).  
(Entity, Subject, Attribute, Value);  
(Role, Entity, Attribute, Value);  
(Kind, Role, Entity, Attribute);  
(Class, Kind, Role, Entity);  
(Flow, Class, Role, Entity);  
(Behavior, Flow, Class, Role);

Messages (Model : Resource) as Resource set specifications. Subject, Attribute, Value : Resource.

Determine class (reified layers contexts) hierarchies:  
(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model, resources, statements, kinds.  
Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch,  
event bus routes. URIs / IDs mappings. Resource set specification resolution. MetaGraph resolves concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics via MetaGraph driven transforms (data / schema / behavior augmentation: dialogs).

Models definition: data (Statement, Entity), schema / contex (Role, Class), interactions / behavior (Flow, Behavior).

Kinds / Roles:  
Grammar: kinds layers aggregation (CSPO layers Kinds).  
Layers: Roles (Models metaclass context resources).  
Reified Kind: (Kind, Occurrence, Attribute, Value);

**Schema Model (Grammars)**

Schema Model layers population / augmentation.

Grammar Resource input set model specificatíon (Statement layer kinds Messages).

Grammars: Predicate Kind from Subject / Object Kind. Kind model layers.

**Behavior Model (Dimensional annotations)**

Dimensional Model layers population / augmentation. Purpose modelling. Dimensional Concepts.

Order layers statements. Hierarchies (contexts / kinds). Parent / child relationships (steps). Order type relationships: husband: single / marriage / married.

(Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.  
(Measure, Value, Previous, Distance);  
(Unit, Measure, Value, Previous);  
(Dimension, Unit, Measure, Value);  
(Concept, Dimension, Unit, Measure);  
(Resource, Concept, Dimension, Unit);  
(Statement, Resource, Concept, Dimension);

Populate / align / annotate models with dimensional data. Model input: statements (model resources). Model specification: augment, sort  
statements. Model specification: specialization of base model layers. Resolve resolution statements order.

Dimensional input set model specificatíon (from Statement layer, ordered SPOs: order criteria, comparisons. Kinds / class / occurrence / instance  
order criteria?).

Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements layers).

Value -> distance(prev, next); ordering;

Assert knowledge: 1h -> 60min;

dom-lun-mar-mie-jue-vie-sab (orders);

1mt -> 100cm;

etc.

Comparison / order: Alignments (prev, curr, next asserted knowledge). Next hour, location, city, country, next distance at next time at current speed. Event sourcing / tracking: married -> marriage occurred.

Sort: cause / effect, temporal, etc. Messages align, functional map, fold, etc. Primitives. Encode layered statements ordering. Complement / supplement concepts definitions.

Events metamodel (TBD):

(Object, State, Axis, Type)  
(State, Axis, Type, Event)  
(Axis, Type, Event, Event)  
(Type, Event, Event, Event)  
(Event, Event, Event, Event)

**MetaGraph Model (models aggregations)**

See Message Resolution.

Model MetaGraph (TBD):

MetaGraph: Resource, Statement, Kind class / instance as CSPO MetaGraph statement roles. Class / subclass relationship, Kind / subkind relationship. Grammar / Model bindings.

Grammar: layers aggregate kinds from resource / statement layer or kinds for each model layers.

Layers, contexts, occurrences, kinds: Role Entity layer occurrences instantiated with each Entity SPO as Entity subject (Entities occurrences in Role context for each Entity SPO). Idem for subsequent layers.

Statement class: context.

Statement instance: context occurrence.

State resource kind in occurrence in context.

State resource (context) class / (occurrence) kind hierarchies.

State Resource URIs occurrences / class IDs.

Resolve Message matching Resource from behavior layers / matching kinds from Model / data layers.

(Kind, SuperKind, Attribute, Value);

(Occurrence, Kind, SuperKind, Attribute);

(Context, Occurrence, Kind, SuperKind); (attributes / links bindings).

(Resource, Context, Occurrence, Kind); State Resource Kind in occurrence context (context / role bindings).

(Statement, Resource, Context, Occurrence); State Resource URIs occurrences / Resource class IDs (classification bindings).

(Interaction, Statement, Resource, Context);

(Action, Interaction, Statement, Resource);

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values.

Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

Ontology matching (table, pk, col, val example). Helper upper models for models linking / alignment.

Events declarative definition. State change of value in axis in measure of context.

Messaging metamodel:

(Message, Resource, LHS, RHS);  
(Interaction, Message, Resource, LHS);  
(Role, Interaction, Message, Resource);  
(Context, Role, Interaction, Message);  
(Dataflow, Context, Role, Interaction);

**Datasources / Backends / Services (URIs)**

TBD.

**Addressing. IDs. Encodings**

Resource<T : URI> monadic hierarchy.

Basic hypermedia browse / CRUD (HTTP verbs) bound Message functors compatible for all Resources (REST).

Resource.flatMap(Message::apply) : Observable<Resource> (stream). Composable functions.

Basic Message application (Context Mapping): shift right mapped applied statement resources. Mapped resource context > instance (occurrence) of next layer message reified resource context.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings. Resource set specification resolution. MetaGraph resolves concrete resources, Message expansion.

Resolve Message / dialog (CRUD) semantics via MetaGraph driven transforms (data / schema / behavior augmentation: dialogs).

Dimensional / Grammar models.

aX^4 + bY^3 + cZ^2 = dW;  
d, a, b, c: classes (CSPO);  
WXYZ: instances (CSPO);  
Powers: CSPO role;  
Terms: CSPO resources;  
Z(obj) is Y(pred) for X(subj) in W(ctx);

Instance, class, metaclass, occurrence terms. Primitives, variables, placeholders.

Resolution (Discovery, DIDs). Templates (grammars). Subjects: attr / val, ctx / role.

Behavior: order / compare.

Proof of work.

MetaGraph model: map URIs -> IDs.

Satisfy dW. Sync resolution (recurse terms contexts).

FCA. Resource attributes.

Tensor, adjacency matrix, tree.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics. Dimensional / Grammar models.

Naming: Context URIs. Dimensional (Statement, Resource, Kind) addressing (conventions). Discovery: patterns / locators: Semantic URIs / MetaGraph ID mappings. Encodings: contextually encoded addresses / URIs.

Naming: NLP. Bind / suggest human readable names / labels.

Naming: Source (plain class) URIs.

Naming: Statement (Context) addresses.

Naming: Occurrence URIs (in Statement in CSPO role).

Naming: Contextually encoded addresses (URIs in Occurrences in Statements in relation with other occurrences).

Naming: Kinds addresses (global / mask, from occurrences in statements). Signatures. MetaGraph: operate over IDs.

TBD.

**Dataflow (reactive models)**

TBD.

For input Resource(s) (Model reactive / async IO APIs):  
. Create / retrieve Model  
. Create / retrieve Context Message(s)  
. Create / retrieve Message(s) Interactions  
. Bind Interaction Message Resource(s)  
. Perform Message transform. Materialize results. Message application rules: upper / domain ontology selectors (closest matching role in  
hierarchies), context alignments.

Match request statement / graph with model via context in interaction (algorithm: addressing, encoding, interaction model upper bindings /  
alignments). Resource MetaGraph. Reified model resources (Resource, Statement, Kinds, CSPO, etc.).

Apply subsequent transforms in interaction context (referrer context, get classes playing entity role, get behavior flows, browse / navigate  
streams). Context, variables, wildcards, placeholders.

Services: distributed addressing / resolution, reactive distributed event bus: streams / contracts, index, naming, registry.

Discovery: All model kinds are browseable / discoverable.

Determine class (reified layers contexts) hierarchies:

(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model resources, statements, kinds.

Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

Message flow (event loop) in / out:

Activation (data) <-> Alignment (schema) <-> Aggregation (behavior);

Encode behavior in statements / graph:  
Comparisons, order. Sort. Order (kinds hierarchy?).

Pattern matching, iteration, jumps. Discovery: routes / signatures, next event in bus / graph.

Context Model Message: Resource Specification (Grammar Template).

Messages Model: context model instance from input model grammar. Transform: context model instance from Messages.

Express Augmentation (Activation, Alignment, Aggregation) as Messages / Transforms. Reified Model entity types / roles (CSPO, Kinds, Layers, etc.).

Resource monad of URIs or Message monad of Resource?

Encoding. Addressing. Schema / MetaModel for data (Model), schema (Context), behavior (Interaction) resources / layers (aggregation). Naming  
formats / schemes: namespaces, contexts.

Class hierarchies (express context / class / kinds hierarchy). Grammars / Dimensional metadata.

Resource MetaGraph bindings (Message expansion / resolution index).

Subscription, Subscriber, Producer, Consumer, Processor. Example: submitting Behavior layer grammar / context "template" initiates "dialog" for fulfill Behavior expanding Message(s) and nested context layer statements (known / resolvable, new behavior / subitems) needed to complete / update full Behavior layers contexts graph.  
Augment. Alignment, Activation, Aggregation Message(s) : Resource set specifications.

Model listens onMessage (interaction context model population / dialogs scopes / namespaces).

Model augments input Message (augmentation specifications over in Message).

Model expands Message (Message over model resources):

Resource listen modelMessage. Model subscribes to response.

Matching triggered Resource. Message matching semantics (transforms).

Triggered Resource publish itself modelMessage.

Model augments output Message (augmentation specifications over out Message).

Model publish onMessage (interaction context model dialogs / resource dumps).

**Messages: Transforms. Graph Execution Semantics**

Message encoding semantics resolve transform execution resource set declaratively from MetaGraph / Models.

Specification resolves to query / create / update / delete according interaction contexts. Messages models determines “possible” messages according models grammars. Interaction specifications (statement / graph / dialog) may have any message encoding components in corresponding statement roles.

For each behavior, flow, class, kind, entity, statement in input request, transforms matches those components by applying messages into model resources (grammar) matched into interaction model (binding subsequent roles by dialogs).

New (potentially unknown) resources are added and augmented into the graph. Augmented resource events emitted from transform streams.

Example: a message composed of a kinds CSPO matches statements “instances” of those specifications (statements whose CSPO have matching kinds). A message with three CSP kinds and a (potentially unknown) object URI retrieves matching resources having that object value into corresponding property kinds. An statement of plain (potentially unknown) URIs instantiates / updates and augments new / known resources added to models and returns an augmentation transform result.

Interaction Model: Context of Messages model for a given interactions session / dialog state. Message invocation requests: Statement(s) building Resource invocation graph with layers matching Message patterns. Layers graph invocation patterns matching from higher to lower layers resources fulfilling higher layers templates. Variables, wildcards, placeholders.

Dialog arguments resolutions example: higher layer Resource / Message request / invocation instantiates in Interaction Transform context corresponding lower layer graph statements to be “populated” to fulfill request. Message IO of “forms” (Messages) inter-peers (originating peer  
acting as “server”) for initial requested peer to “ask” for form elements to be populated (interaction context “dialogs”). Resolution may propagate to other peers (content aware addressing dataflow routes dispatch: P2P resources address encodings, matching forms models requests). Nested interactions.

Explain messages (resource resolution). Grammar. Match model Resource(s). Compound nested CSPO statement contexts defines result behaviors. Message CSPO contexts may define create, retrieve, update or delete operations (passing 'null' for example for resource / statement to be deleted).

Explain transforms (message application). Transform: Resource stream result of Message application over resolved Resource(s)). Input statements: Message(s) / Resource(s) (from input message or to be populated or populated in dialog) and "goal" Message / Resource aggregating a model from Resource MetaGraph with Message / Resource bindings.

Message types (Augmentation: onto / domains):  
Attribute / Link (data):  
. Alignment: Augment / infer Attribute / Link.  
Class / ID (schema):  
. Activation: Augment / infer Kind, Class.  
Role / Context (behavior):  
. Aggregation: Augment / infer Role / Context.

Runtime / Resources / Messages: Core (upper / onto) Resources, Messages, Transforms. Reified entities (CSPO, Kind, SubjectKind, etc.). Match cases in messages.

Core (upper / onto) Messages: Getters, setters, nav, etc.

Domain Messages: raiseSal: setSal(sal \* increment); promotion: setPosition.

Event sourcing / tracking: married -> marriage occurred.

Resource.flatMap(messageInst::apply) : Resource.

Dataflow: Messages hierarchy. Aggregate contexts from coarse to fine grained transforms (raiseSal -> setAttr).  
data <-> schema <-> behavior.

Message dispatch, input statements resolve to applicable messages from switch from behavior to data layer invoking async microservice.  
Message case matching may involve entering and leaving data, schema and behavior paths if aggregated contexts matches more than one  
message. Visitor.

Message: functor (monadic transform) : Resource<T> -> R, T, R : URIs (hierarchies, models, semantic content types). Available verbs / flows /  
navigation (browse models, state of application returned from materialized models). Parameterized functions (partial applications) into Messages metamodel resources. Contexts (dataflow). Execution graph.

Alignment Message: Resource -> Statements (attributes, values).  
Activation Message: Statement -> Kind, Class.  
Aggregation Message: Statement -> Statement (next layer).

Subscriptions declarations / definitions. Applied on streams activations (transforms, executions resource parameterized partial contexts).

Messages metamodel: functor declarations partially defined over metamodels resource (T) defining transforms into (R) over application  
(flatMap) over / into (S). Messages inferred / aligned, activated, aggregated according base message transforms resources. Messages inferred from models / layers. TBD.

Functors <T, R> -> Resource<R>

Form / Template describing (reified as a Resource in a context model) declaratively subscriptions and actual exchange capabilities (datflow).  
Mappings, Transforms.

Processor which acts upon Resource events. Materialize results.

Specify declaratively augmentations by means of messages.

Upper onto / domain aggregated messages.

Event bus: P2P deployment.

Messages: Monadic applicables over Resource (flatMap).

Base HTTP / Browse (REST) Messages. Custom Messages.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings. Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

**Augmentation (via Messages)**

Activation (Statement / Entities : data).

Alignment (Kinds / Classes : context / schema).

Aggregation (Flows / Behaviors : interaction).

Messages describes declaratively augmentation steps materializing models contexts / hierarchy layers.

**Protocol (API): dialogs (distributed resource augmentation / sync)**

Message resolution (contexts).

Reactive. Interaction / session contexts.

Annotate, link, browse resources instances, classes, metaclasses, occurrences in roles in contexts, attributes / values. Services / clients: endpoints: Virtualization (wrapper protocols).

Semantically annotated content types: image/png;face, text/xml;faceImgCoords. RDF schemas describing content, attributes, links in context / target roles. Content types: labels (schemas).

Message: Context Model API. Input statements: Model Grammar. Augmented IO by interaction transforms of applied matching Message with model statements inputs. Context of core models instances. API.

Transform: Interaction Model API. Input statements: Transform request invocation specification. Functional application of Message(s) over Resource(s): Transform (streams). Augmented IO: Requested Transform which applied augments resulting responses (dialog arguments  
resolutions). Context of context model instances.

Reactive / streams API. Message Transform (interaction result): matches request context specification built upon Resources / Messages (TransformBuilder). Resolve state / dialog session graph. Returns observable stream. Dataflow (chaining). Operations (over streams).

Transform request invocation specifications: means to interact with underlying contexts models (CRUD, domains behavior). Transforms result from applicating Message(s) over Resource(s). Sending a Message Resource to a given interaction context initiates a “dialog” in which to “populate” target Resource(s) and Resource arguments. Each dialog “step” renders resources / layers streams of requested arguments (server “queries” clients) or resources / layers streams of response augmented Resource(s).

Graph linking / alignment / synchronization by entailments from event sourcing over inferred state. Distributed predictive alignments.

DOM / OGM APIs (JAF). I/O Implementation, Deployment.  
Model, URI, Resource, Statement, Kind hierarchies. Models architecture (URI class per layer). DIDs / P2P / Rx Implementations. Model API. ModelManager. Event loop. IO.

**Protocol (API): resource activation (hypermedia application browser)**

Reactive. Interaction / session contexts.

Protocols / Services / Clients: Context interaction sessions (state flows).

Content type activation. Messages / gestures. Rules (commands / verbs). Content types: labels (schemas).

Browser referring context (Work, Peter, Employee).

Annotations (protocol): JSON-LD. Model / Grammar / Dimensional. Map annotations to resources (query string / meta resource description). Browse data (model), schema (grammar), behavior (metagraph).

Models ‘plug’ into Runtime augmenting its capabilities via standard extension APIs (added features / knowledge reactive URIs). Models ‘modules’: parsing modules declarative descriptions. Augment, link instance data.

Upper aligned ontology plugins / blueprints:

Resource URIs specialized implementations for different connectors / endpoints and content types (DB / OData, REST / HAL, etc.). Feature  
Resources backends (i.e.: URI for DB interaction).  
Purposes: Metamodel declarative goal statement. Fulfill flows (templates / forms: Messages).

Goal: P2P service that connects to services / endpoints (DB, REST, etc.), homogenizes them and exposes an API by which (augmented)  
knowledge of an stated entity is returned in response (protocol that entails queries / CRUD, object navigation in message / session state contexts). Peer shares / syncs with other peers.

Goal: Intermediate API (HAL for example) aggregating previous objects knowledge (DCI, DOM, OGM, MVC)

Goal: Semantic Browser. Homogenize diverse domains. Query examples. Search session history. Referrer semantics. Collected items in goals roles. Create session purpose document. Link to / from any addressable resource in context / role. Annotate source / destination context roles,  
attributes and schema.

TBD.

**Ontology matching**

TBD.

**Data / Reference Model (APIs, Functional Semantics)**

TBD.

Upper ontology: Node "levels" of domains abstraction. Highest level: service / user interaction (resource / hypermedia activation: model gestures). Lowest levels: upper ontology / business domains.

Application / Site / Service node types (Node ontologies domains layers). Renderers producers / consumers. Backends integration (Augmentation, Messages).

**Platform: implementation**

Introduction. Document. Use Cases (EHR). Standards. Models (predictions / signatures).

Implementation. Languages. Backends. Reactive frameworks / microservices. Distributed consistency. P2P / DIDs. Models / APIs. Nodes / Endpoints. Containers. Deployment.

Implementation: render RDFS / OWL upper ontology aligned (sameAs, type, subClassOf, restrictions, etc.). from Model / Message+ XSLT transforms. Semantic engine / reasoner / backend (URI published reactive service, Message based wrapper). Record Model / Message transforms.

API: URI, Resource, Message, Statement, Kind, Layers. Representation: XML bindings.

Kind : Statement : Message : Resource : URI;

URI<T extends URI> : Monad.

Resource: (URI, URI, URI, URI);

Message: specification / transform (input / output dialog).

XSLT / XPath / XLink / XPointer / XQuery.

Resource XML Encoding (nested layers quads).

Message XML Encoding.

XSLT templates (Resolution, Activation, Alignment, Aggregation). Resolution algorithm: TBD (ontology matching).

Events: Dataflow. Reactive Model endpoint Message dispatch / resolution (Producer). Resolve (addressable) Message resources (Resolution template). Apply templates (Resolved resources : model / Message resources : view context) : XML (Message).

Ontology levels: data / schema / behavior (backend, business, frontend) objects.

TBD.

**Application**

Products And Services Community Exchange Network:

Going through my most recent attempts of having something concrete for sharing in plain English I realize one mistake I'm committing: I'm trying to describe combustion vehicles (Hypermedia Applications) saying that petroleum exists (Semantic Intelligence).  
  
As long as my post are going I've just got a stack of (incoherent) "analysis" documents as the result of my work. And I had only those until now because I was stuck because of the previously mentioned mistake (ah, and because of my Bipolar Disease maniac episodes...).  
  
I should try to describe applications instead and see how and where fuel should burn properly inside a motion vehicle to generate traction. Every semicolon I write is updated into my GitHub repository, so, sorry if you browse that "scrapbook" and you don't find anything even intelligible.  
  
First, I'll try to describe a "problem" (problem "spaces" in this case) and how a Purpose driven user Community achieves its Goal(s) by means of Goods, Products and Needs satisfaction (ontology levels: from abstract upper ontology to user gesture command in user interface / service invocation).  
  
The problem is to organize interdisciplinary (multiple domains) Task(s) in a Purpose fulfilment network with Actors, Contexts and Roles (with attributes and values). Problem spaces (domains) are declaratively stated by DCI[1] design pattern: Data / Context / Interaction use cases definitions and instances.  
  
Collaborative Federated Actor network complying determinate Profile(s) satisfying specific Product / Good / Need abstraction playing determinate Role in use cases Context.  
  
Domain Translation between business domains, example: orders, delivery, invoicing (micro) services Model instances are the means by which distributed disparate data, schema and behavior of different sources (applications, services) integration could be performed by means of Semantic Intelligence and Augmentation Protocol(s).  
  
A domain can be defined in terms of a set of actions / tasks with the Purpose of satisfying some Goal solving the Need for a Good producing / gathering a Product. Ontology. Purpose as Goal “class”.

The principal focus is to deploy a (social) Collaborative peer (Actor) network for which entities and individuals develop Profile(s) which acquaint them with Purpose resolution capabilities. Then, according peer’s specific needs (domain Goals) the application orchestrates interactions needed for Product(s) Task(s) accomplishment.

Ontology:

Domain / Actor / Context / Role / Product / Good / Need / Purpose / Task / Goal / Exchange.

Domains: data, schema and behavior of business applications (ERP, CRM, BI, SCM, HMS, etc.).

General purpose business domains problem resolution / tasks, goals accomplishment helper tools.

Syndication (contextual hypermedia activation): QA. Polls. Learning. Profiles. Guided task (wizards), guided editors: Context: Goal / Purpose.

Contents: Wiki view of augmented knowledge. Addressing. Hypermedia. API (Wiki) render nodes / links semantically browseable.

Backend: Nodes / Protocol.  
  
SoLiD:

<https://solid.mit.edu>

DIDs (Blockchain dApps):

<https://w3c-ccg.github.io/did-spec/>

<https://ont.io/#/>

Executable models (flows): testing results, prompts, scoring.

Applications (use / implement like):

Drive / Jira / Trello / Keep / Mural / Tasks / Calendar.  
  
Ontology levels abstractions (data, schema, behavior): service / user interface rendering (activation).   
  
Dashboards components (widgets / media / extended content types / addressing).  
  
Hypermedia Activation. Addressing. Link extended content types resources elements / parts with other resources addressed elements.

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.  
  
Distributed P2P (Blockchain) approach of data synchronization between peers for ease of deployment patterns election and datasources integration (APIs, microservices, etc.).  
  
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, equivalent attributes, equivalent roles).  
  
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).

Alignments Augmentations:  
  
Activation: type inference : classification (determine class / metaclass / roles for entity attributes and values).  
  
Activation infer attributes / relations : clustering (from multiple occurrences of same entity in diverse data sources).  
  
Aggregation: infer roles in contexts: regression (Person class in Employment interaction : Developer role).  
  
Integration of addressable resources. Reactive I/O (sync back ends). Content type driven semantic augmentation / annotations.

Augmentation of distributed resources. Annotations (Semantic / ML). API for resource / schema / interactions exploration / protocol for message based API "dialogs" execution. HAL (Hypertext Application Language), OData (REST) like interfaces.  
  
Example: Google Drive / Google Knowledge Graph APIs Augmented with ML / Semantic intelligence tailored for specific domains / application kinds.

Augmentation. Ontology matching. Hypermedia augmentation protocol. Browser / Client APIs.

URIs API for annotating network retrieveable resources metadata. Content type / model driven augmentations / activations (models features / outputs). Subject attributes / values. Occurrences contexts / roles. Paths, pointers, locators. Example: annotate document URIs (parts, sections, mentions), annotate images URI (whole image description, coords: classes, individuals), annotate DB, table, row, column, value URIs, annotate / describe service / APIs URIs. Hypermedia protocol composable with other (described / annotated) APIs / resources. Example: Drive APIs.

What my attempts are about where, in the beginning, to match different URIs or 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.  
  
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);  
  
Each layer abstract:  
  
Statement (data instance):   
(Statement, Occurrence, Attribute, Value);  
someOne buys someProduct  
  
Entity (data class):  
(Entity, Statement, Occurrence, Attribute);  
someBuyer, someProduct (Entity);  
  
Role (schema instance):  
(Role, Entity, Statement, Occurrence);  
Buyer, Product (Role);  
  
Class (schema class):  
(Class, Role, Entity, Statement);  
Person, Good (Class);  
  
Flow (behavior instance):  
(Flow, Class, Role, Entity);  
someBought (Flow);  
  
Behavior (behavior class):  
(Behavior, Flow, Class, Role);  
Buy (Behavior);  
  
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 metamodel. The act of applying an Augmentation implies one source Resource (context), one template Resource (transform) and a resulting (set of) Resource(s).  
  
One also could Augment Resource(s) in a functional manner, using reactive event driven APIs so, for example applying "Person" class to "Employee" role could shield a Resource set of people being working for someone. The ultimate goal is to be able to "plug" as much "backends" connectors as posible into distributed peers which exposes protocols / APIs for knowledge driven hypermedia applications.

Implementation. Async / Reactive Service URIs / Connectors (sample):

Spark,

Lucene / Solr,

Kafka,

MQ,

ServiceMix,

Vert.x,

OSGi,

Spring Boot,

SCDF,

Jena, (RDFS, OWL, Turtle, N3, SPARQL),

Reasoning / Shapes,

JAF / JCA / JDBC / JNDI / JMX,

Metamodel / Teiid / D2RQ / OData,

HAL,

DIDs,

OpenShift (containers / deployment).

Application:

features / techniques / patterns.

Implementation deployment use cases. Sample Apps: SoLiD / PIM / PASCEN: App declaratively built with framework, Implementation Integrations.

[1] <https://en.wikipedia.org/wiki/Data,_context_and_interaction>