# Hello, World!

I'm Sebastian Samaruga, software developer from Buenos Aires, Argentina. I'm currently mildly seasoned in the development of business applications in the Java platform and related technologies and stacks. But what has got me scratching my head a lot in the last couple of years is the "Semantic Web". I know it is a technology paradigm far away from it's realization. But, perhaps, with the advent of Big Data and such it could finally find its niche. Meanwhile I dump all my head scratching thoughts into this blog and a related GitHub repository I'd like to share. It is an endless work in progress draft and a scrapbook of spare assorted ideas waiting to be realized...

[Creative Commons License](http://creativecommons.org/licenses/by-nc-sa/4.0/)  
This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-nc-sa/4.0/).

Introduction: <https://github.com/sebxama/scrapbook/raw/master/Items.docx>

Sebastian Samaruga

[https://sebxama.blogspot.com](https://sebxama.blogspot.com/)

**Items:**

**Objectives:**

Rosetta Stone like engine enabling Ontology (Data, Schema / Information, Knowledge / Behavior) discovery, matching and integration.

Reactive Service Bus for pluggable integration of application and translation of gestures between domains business systems allowing workflows alignment and discovery of application systems behavior.

**Layers: RDF Quads Representation. Augmentation / Inference Matrix Models**

Patterns:

(Context, Occurrence, Attribute, Value);

(Dimension, Measure, Unit, Value);

(Dimension, Resource : SPO, Kind, Statement);

Sets Model Layers Structure:

(Dimension, Resource, Kind, Statement);

(Statement, Dimension, Resource, Kind);

(Kind, Statement, Dimension, Resource);

(Resource, Kind, Statement, Dimension);

(Dimension, Resource, Kind, Statement);

Dimension: U

Resource: SPO

Kind: SPO Intersections (pairs)

Subject Kind: P intersection O

Predicate Kind: S intersection O

Object Kind: S intersection P

Statement: SPO Intersection (of the three sets)

Sets / Individuals Mappings:

IDs: metaclass, class, instance, context, role, occurrence, previous, next ID roles relations for Model Set Contexts.

Augmentations / Transforms: Model / Domains functional mappings. Order. Dimensions. Axes. Flows. Hierarchies. Inference / Population.

Levels: Augmentations. Mappings.

Levels: Resource, Kind, Statement.

Levels: Reify Statement as Kind, Kind as Resource, Resource as Statement.

Levels: Reify Resource as Kind, Kind as Statement, Statement as Resource.

Sets / Individuals Mappings:

Levels (layer statements) shifts (quads matrix). CSPO roles:

(Dimension, Resource, Kind, Statement);

Resource: Semantic Address / ID. Dereference / Browse: Representation State / Roles (CSPO) according referrer / occuurrence / transforms / mappings (monadic pipelines) contexts. Examples: Faceted search (build context template schema roles), Shopping Cart (build use case representation items via state representation exchanges).

**Models: Sets, Individuals, Mappings**

Models which are instances of the Sets Layers Model Structure. Model Properties:

Metaclass, Class, Instance, Occurrence, Context, Role, Attribute, Value.

Functional. Mappings / Transforms. T-Box / A-Box. Sets, Groups, Categories: TBD.

Types Model:

Types (types in sets roles):

(Relation : Statement, Relationship : Kind, Role : SPO, Dimension : U);

(Dimension, Context, Class, Resource);

(Resource, Dimension, Context, Class);

(Class, Resource, Dimension, Context);

(Context : Statement, Class : Kind, Resource : SPO, Dimension : U);

Individuals Model:

Individuals (individuals / sets types instances):

(Context : Statement, Class : Kind, Resource : SPO, Dimension : U);

(Dimension, Context, Class, Resource);

(Resource, Dimension, Context, Class);

(Class, Resource, Dimension, Context);

(Context : Statement, Class : Kind, Resource : SPO, Dimension : U);

Mappings Model:

Mappings (type / individual relationships):

(Context : Dimension, Occurrence : Measure / SPO, Attribute : Unit / Kind, Value : Value / Statement);

Models metadata, properties and upper alignments / augmentations relationships Model data.

(Value, Context, Occurrence, Attribute);

(Attribute, Value, Context, Occurrence);

(Occurrence, Attribute, Value, Context);

(Context, Occurrence, Attribute, Value);

**Layers: Augmentations / Inference**

Activation: Classification (Context types Occurrences Attributes).

Activation: Which Attributes has Context Occurrence (according to its Kind in Context / Role) in this Occurrence.

Alignment: Regression (Context types Occurrences Attributes Values).

Alignment: Context Occurrence Attributes Values (according to its Kind in Context / Role).

Aggregation: Clustering (Context types Occurrences).

Aggregation: Context type instance aggregates type instance child Occurrences (parent Context type instances) matching grouping criteria (Encoding).

Augmentations:

(Context, Occurrence) : Value;

Activation:

(Statement, Resource) : Kind;

Alignment:

(Kind, Statement) : Resource;

Aggregation:

(Resource, Kind) : Statement;

**Model Semantics:**

Data: Individuals. Mappings. Data Occurrences Aggregation.

Data: Individuals Model.

(Dimension, Context, Class, Resource);

Mappings (type / individual relationships):

(Context : Dimension, Occurrence : Measure / SPO, Attribute : Unit / Kind, Value : Value / Statement);

Information: Types. Mappings. Type Occurrences Attributes. Activation.

Information: Types Model. Schema.

(Dimension, Relation, Relationship, Role);

Mappings (type / individual relationships):

(Context : Dimension, Occurrence : Measure / SPO, Attribute : Unit / Kind, Value : Value / Statement);

Knowledge: Individuals / Types Mappings (Attributes) Values. Alignment.

Knowledge: Behaviors.

Mappings (type / individual relationships):

(Context : Dimension, Occurrence : Measure / SPO, Attribute : Unit / Kind, Value : Value / Statement);

**Ontology Matching: Relations / Relationships**

Entity Relationship instance asserted as a reified concept with its type and attributes or as a series of triple statements which describes the given Entity Relationship instance via individual assertions. Bidirectional translation.

aPerson loves anotherPerson.

Person loverOf Person.

loverOf predicate: Kind of aPerson. Domain / Range. Dataflow (Functional Augmentations).

Loving: loverOf Kind.

aLoving: loves Kind.

TBD: Relationship / Relation

Reify Kinds as SPOs : Types Model

Reify Statements as / Kinds / SPOs : Mappings Model

Augmentations (Aggregation).

**Ontology Matching: Dimensional Alignments (Mappings):**

Explain Layer Context, Occurrence, Attribute, Value Pattern for Models SPO Statements functional mappings expansion:

(Context, Occurrence, Attribute, Value);

For a given CSPO Quad:

(C, S, P, O);

Expansion:

(C, P, S, O);

(C, O, P, S);

TBD.

Mappings (set / individual relationships):

(Context : Dimension, Occurrence : Measure, Attribute : Unit, Value : Value);

Order. Comparison. Relations. Upper Ontology assertions. Augmentations. TBD.

Relation / Relationship: Tabular / OGM (Object Graph Mapper):

I/O: (Class, ClassID, Attribute, Value);

Class: Table / Object Type.

ClassID: PK / Object ID.

Attribute: Column / Member.

Value: Cell / Field Value.

Subject Kind: Relation / Domain.

Predicate Kind: Relationship.

Object Kind: Mapping / Range.

Dataflow: Reactive Functional Augmentation / Integration APIs.

Indices: Apply functional mappings expansion.

**Functional API: Monads / Transforms**

Resource / Layer?

Context / Occurrence / Mapping?

Mapping: Selector Monad. Matching Resource / Role set?

Context / Occurrence Monads wrapping Layers Hierarchy Contexts.

Entity Alignment / Matching resolution via Functional Augmentations: Agggregations / Activation / Alignments (upper / dimensional matchings). Versioned graph: stateless / functional. Mappings assertions matching.

APIs: Augmentations, Query, Traversal, Matching, Transforms. Functional APIs Query / Browse / Traversal / Transforms examples. Encoding / Matching.

**Encoding: Functional Mappings**

Masks: Predicates of Set memberships. Functional Mappings. ID encoded state / transforms. Models merge. Ontology Matching. Mappings Model: Types / Instances Models merge (upper) Augmentations.

Mappings / Functional Encoding: Upper Dimensional Matchings / Augmentations. Mappings Model masks matchings reflects / leads to Types / Individuals Models Augmentations / Assertions.

Mappings / Functional Encoding: Relation Statements / Relationship views / matchings examples.

Mappings Upper Alignments examples (dates, marital status, hiring). Relation Relationship statements order / context properties (Dimensional Alignments).

IDs: metaclass, class, instance, context, role, occurrence, previous, next ID roles relations for Model Set Contexts.

Augmentations / Transforms: Model / Domains functional mappings. Order. Dimensions. Axes. Flows. Hierarchies. Inference / Population.

Levels: Augmentations. Mappings.

Levels: Resource, Kind, Statement.

Levels: Reify Statement as Kind, Kind as Resource, Resource as Statement.

Levels: Reify Resource as Kind, Kind as Statement, Statement as Resource.

Resource: Semantic Address / ID. Dereference / Browse: Representation State / Roles (CSPO) according referrer / occuurrence / transforms / mappings (monadic pipelines) contexts. Examples: Faceted search (build context template schema roles), Shopping Cart (build use case representation items via state representation exchanges).

**TODO Items:**

* Dimensions Encoding: Given Dimensional Contexts (CSPO Models set layouts) having four dimensional sets (Types Model, Individuals Model, Mappings Model, State Model) each representing (nested) CSPO inputs / parts of a recursively aggregated CSPO layout (i.e. aggregated layout Context is Mappings Model, Subject is State Model, etc.) having this setting (Models types / layers class / instance IDs) reified in this fifth "Focus" Model which represents a "snapshot" of current state and available transitions (Focus shifts).
* Models: CSPO Layers (matrix) layout.
* Focus Mapping Model. Axes (X / Y: Model instances matrices, cycles), intersection (Z: Model instance matrix):
* Model patterns:
* (Dimension, Unit, Measure, Value);
* (Context, Occurrence, Attribute, Value);
* Context / Dimension / Context:
* Occurrence / Unit / Subject:
* Attribute / Measure / Predicate:
* Value / Value / Object:
* X Model: Context / Schema / Information / Relationships
* Y Model: Data / Relations
* Z Model: Interaction / Context instance Data state calculated intersection. Behavior
* Upper Y / Lower Y: Previous / next data state.
* Left X / Right X: Previous / next context state.
* Augmentations calculate current, previous, next Model states.
* ImplementationItems & drafts documents topics.
* Diagrams (TO DO):

**ImplementationItems:**

**Meta Model:**

Meta Model: Layered class hierarchy of RDF Statement (Quads) stacked Layers Mappings. Model shapes / grammar metadata. Augmentations reify to / from meta model mappings / CSPO ontology statements.

Layers: URI, Instance, Class, Context, Metaclass, Occurrence, Statement, Relation, Relationship shape mappings (layers transforms / message augmentation templates) grammar shapes.

Model Layers rendered into different APIs / Representations (RDF4J SAIL Layers):

RDF Store SAIL:

Quads Layers / Objects Mappings Meta Model Metadata Encoding: RDF / RDFS Ontology (Upper / Schema). Meta Model Reified Order / Matching Encoding.

Object / Graph Mapper (OGM) SAIL:

Object Mappings: Singleton POJOs mapping Quads Statements plus helper methods (aggregate context subjects, subject attributes, etc.). Quads / Functional IO. Layers Object Mapping (Meta Model metadata populated).

Functional APIs SAIL:

Functional Layers Wrapper: Augmentations, Metadata, Service Transforms Namespaces. Meta Model Reified Functions / Mappings Encoding.

Facades SAIL:

Business Layer: DCI Ontology Models for Forms / Flows (HAL / HATEOAS) Use Cases rendering: Declarative REST / WS-\* Endpoint Templates. Application Ontology aligned Models.

Layers specs. All SAILs renders in some form the following data model / schema:

URI

(URI, URI, URI, URI)

Instance : URI

(Instance, URI, URI, URI)

Class : Instance

(Class, Instance, URI, URI)

Context : Class

(Context, Class, Instance, URI)

Metaclass (Kind) : Context

(Metaclass, Context, Class, Instance)

Occurrence : Metaclass

(Occurrence, Metaclass, Context, Class)

Statement (Entity) : Occurrence

(Statement, Occurrence, Metaclass, Context)

Relation : Statement

(Relation, Statement, Occurrence, Metaclass)

Relationship : Relation

(Relationship, Relation, Statement, Occurrence)

Aligned Statements. Entity ontology matching / alignment. Multiple Statement Occurrences. TBD.

Augmentations:

Layers Aggregation:

Determine (Layers) which Subjects (Occurrences, aMarriage : Relation) belongs / aggregated into which Statements (Contexts, marriage : Relationship).

Layers Activation:

Determine Occurrence Attributes.

Layers Alignment:

Determine Occurrence Attributes Values.

Meta Model:

Map Layers to CSPO inter / intra Roles. Model / Meta Model Mappings.

Message (SAIL Statements IO) to / from Meta Model:

(C, S, P, O);

to / from:

(Context / Statement, Object / Occurrence, Attribute / Metaclass, Value / Context); (Statement, Relation, Relationship).

Sets / Individuals Mappings:

IDs: metaclass, class, instance, context, role, occurrence, previous, next ID roles relations for Model Set Contexts.

Augmentations / Transforms: Model / Domains functional mappings. Order. Dimensions. Axes. Flows. Hierarchies. Inference / Population.

Levels: Augmentations. Mappings.

Levels: Resource, Kind, Statement.

Levels: Reify Statement as Kind, Kind as Resource, Resource as Statement.

Levels: Reify Resource as Kind, Kind as Statement, Statement as Resource.

Sets / Individuals Mappings:

Levels (layer statements) shifts (quads matrix). CSPO roles:

(Dimension, Resource, Kind, Statement);

Resource: Semantic Address / ID. Dereference / Browse: Representation State / Roles (CSPO) according referrer / occuurrence / transforms / mappings (monadic pipelines) contexts. Examples: Faceted search (build context template schema roles), Shopping Cart (build use case representation items via state representation exchanges).

Events Flow:

URI Layer: Input Layer. Message populates URI Layer.

URI Layer: Augmentations Layers reified (levels). Augmentations Performed.

URI Layer: Outputs materialized URI Statements from query Augmentations final Relationships. Return matching / augmented Message statements.

Layers Meta Model schema:

RDF Store SAIL.

TBD.

Objects Layers Mappings:

Object / Graph Mapper (OGM) SAIL.

TBD.

Functional Layers Mappings:

Functional APIs SAIL. Perform Augmentations / Functional Transforms.

Functional Dataflow.

Layer Monad. Augments Object Mapping for Functional Transforms.

Functional Transforms (Augmentations, Metadata, Services namespaces).

Message : Set<Statement : URI>.

Layer<Set<URI>>

Layer<Set<Instance>>

Layer<Set<Class>>

Layer<Set<Context>>

Layer<Set<Metaclass>>

Layer<Set<Occurrence>>

Layer<Set<Statement>>

Layer<Set<Relation>>

Layer<Set<Relationship>>

Transforms:

For each dataflow bindings (parameterized types):

Augmentations::Aggregate<T, U>

Augmentations::Activate<U, V>

Augmentations::Align<V, W> : Materialized Augmentation Statements.

Metadata namespace: Model Browsing Transform Functions:

Metadata::URIs<T, U>

Metadata::Instances<T, U>

Metadata::Classes<T, U>

Metadata::Contexts<T, U>

Metadata::Metaclasses<T, U>

Metadata::Occurrences<T, U>

Metadata::Statements<T, U>

Metadata::Relations<T, U>

Metadata::Relationships<T, U>

Metadata transforms (browse: order / filter / traversal). Example: Filter, obtain Address Kind Occurrence in Person Statement Entity.

Layer<Set<Statement>> aPerson = Layer.of(personStmts);

aPerson.flatMap(Metadata::Metaclasses).stream().filter(addressKindPredicate).flatMap(Metadata::Occurrences).stream();

TBD.

Services Context Namespaces:

Function / Mapping / Service IO resolver Transform (reactive / events: dataflows). Resolve / Augment Layer occurrences. Context Resolution / matching: Shapes / Zippers (Monads).

Services::Resolve<T, U>

TBD.

Order:

Meta Model: Layers Instances Order. Metadata browse / traversal transforms results order (in context). Layer Alignment. URIs Comparator. Metadata transform (example):

(Instance, URI : Ctx, URI : LT, URI : GT)

TBD.

Matching / Alignment:

TBD.

TBD: Order / Matching / Mappings / Functions / Business: Meta Model Encoded Upper Dimensional Model. Declarative. Templates. Shapes. Addressing (Zippers).

Example:

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

Dimension:

Unit:

Measure : Event (Previous Value state). Mapping / Function argument.

Value: Current Value state (Previous Measure). Mapping / Function result.

Equals: (Measure, Value) -> (Value, Measure).

Examples:

(Time, DayOfWeek, Sunday, Monday);

(Time, DayOfWeek, Monday, Tuesday);

(Time, DayOfWeek, Tuesday, Wednesday);

(Time, DayOfWeek, Wednesday, Thursday);

(Time, DayOfWeek, Thursday, Friday);

(Time, DayOfWeek, Friday, Saturday);

Mappings for: Today, Yesterday, Tomorrow. TBD. DayOfWeek: Ordering Relation Mapping / Function ontology kind.

(Employment, Status, Unemployed, Employee);

(Employment, Status, Employee, Retired);

Mappings for Hiring / Retirement. TBD.

Resolver Service: Resolver.resolve(URI : Layer encoded DIDs) : Aligned, dereferenceable Resource (URI : Resource). Index (VSM / FCA), Naming (Addressing), Registry (Hierarchical Key / Value) Alignment Services.

Ontology Matching: OntResource. DIDs. Blockchain (updates): Endpoints (LOD / sameAs like context resources / arcs assertions / metadata). Augmentation transactions (Resource occurrences blocks). Smart Contracts / Behavior (REST / HAL DCI APIs) Metadata.

To Do: Aggregate SPO statements into (to / from) Relation / Relationship via metadata properties. Order alignment: dimensional / reified schema. Upper alignments: master / detail, cause / effect, temporal, etc.

Resources: Mappings (Relations / Functions from Resource occurrences state) Monads / Transforms: Query / Traversal.

Mappings Properties (Resource occurrences state): metaclass, class, instance, occurrence, role, context metadata properties.

Sets Aggregation: Contexts. Kinds, Resources Statement Monad.

Monad::flatMap(Monad.value);

Context Mappings: occurrence / context metadata:

Statement.of(Context);

* Occurrence: Mapping.
* Context: Attribute / Value (single map argument: retrieve values from occurrence resource mapping).
* S::flatMap(P, O) : C;
* P::flatMap(S, O) : C;
* O::flatMap(S, P) : C;

Kinds Mappings: metaclass / class / role metadata:

Statement.of(Kind);

* Role: Mapping.
* Class: Context.
* Metaclass: Kind.
* S::flatMap(C) : SK;
* P::flatMap(C) : PK;
* O::flatMap(C) : OK;

Resource Mappings: instance metadata:

Statement.of(URI); Layers Resource hierarchy.

* Instance: Mapping.
* Attribute / Value Mappings.
* S::flatMap(P) : O;
* S::flatMap(O) : P;
* P::flatMap(S) : O;
* P::flatMap(O) : S;
* O::flatMap(S) : P;
* O::flatMap(P) : S;
* Context(Kind) : Resource;
* Context(Resource) : Kind;
* Resource(Kind) : Context;
* Resource(Context) : Kind;
* Kind(Context) : Resource;
* Kind(Resource) : Context;

Example (occurrences navigation)::

Kind(Resource) : Context; Kind occurrences coming from previous Resource (navigation context).

Sets, Groups, Categories:

Sets: Contexts, Kinds, Roles, Dimensions.

Individuals: Relations, Relationships, Resources, Dimensions.

OntResource Monad:

Aligned Entities / Resolver Service.

OntResource<Dimension>;

OntResource<Role / Resource>;

OntResource<Relationship / Kind>;

OntResource<Relation / Context>;

Occurrences:

Resource<?>.flatMap(Resource::occurrences) : Resource<Set<Statement>>;

Statement: Context / Relation Dimensional Mappings / Alignments Statements.

Statement Monad: Occurrences

Statement.of(Contexts, Kinds, Roles, Dimensions);

Statement.of(Relations, Relationships, Resources, Dimensions);

Statement.of(Types: Dimension, Role / Resource, Relationship / Kind, Relation / Context) : Set<Statement<Type>>;

Statement::occurrences

Dimension occurrences: Resources.

Resource occurrences: Kinds.

Kind occurrences: Contexts.

(Dimension, Resource, Kind, Context);

Example:

Kind(Resource) : Context; Kind occurrences coming from previous Resource (navigation context).

Dimension occurrences: Roles.

Role occurrences: Relationships.

Relationship occurrences: Relations.

(Dimension, Role, Relationship, Relation);

Mappings / Alignments:

Dimension: Layers. Hierarchy. Encoding (sets / instances). Types. Meta Model hierarchies. Encode / align Sets / Individuals: TBD.

(Dimension, Role, Resource);

(Dimension, Kind, Relationship);

(Dimension, Context, Relation);

(Dimension, Role : Measure, Relationship : Unit, Relation : Value);

(Dimension, Resource : Measure, Kind : Unit, Context : Value);

Mappings / Transforms / Relations / Matching: Dimensional reified. Alignments (upper / matching / order / aggregation / comparison metadata).

Dimensional mappings / navigation context for occurrences : Dimension mappings hierarchy root.

Resource (URN) / Role (CSPO).

Relationship (Employment) / Kind (employee / employer).

Relation (anEmployment) / Context (salary / position).

Relation values in Contexts (roles): Data / Interactions. Sets, FCA, VSM. Base SAIL Layer. OGM, Functional Layers.

Augmentations:

Contexts: sets, Occurrences: individuals, Mappings: Contexts Occurrences.

Aggregation: Clustering (Types).

Activation: Classification (Attributes in Type Context).

Alignment: Regression (Values in Attributes Context).

Contexts (sets):

(Relation, Relationship, Role, Dimension);

Occurrences (individuals):

(Context, Kind, Resource, Dimension);

Mappings (set / individual relationships):

(Context : Dimension, Occurrence : Measure, Attribute : Unit, Value : Value);

Contexts, Occurrences, Mappings: Quads layers hierarchies. Augmentations: Aggregation, Activation, Alignment. Reification: roles / polymorphism (layers contexts type hierarchy).

Contexts:

Dimension : Mapping Context

Role, Dimension : Mapping Occurrence

Relationship, Role, Dimension : Mapping Attribute

Relation, Relationship, Role, Dimension : Mapping Value

Occurrences:

Dimension : Mapping Context

Resource, Dimension : Mapping Occurrence

Kind, Resource, Dimension : Mapping Attribute

Context, Kind, Resource, Dimension : Mapping Value

Mappings:

Value

Attribute, Value

Occurrence, Attribute, Value

Context, Occurrence, Attribute, Value

Mapping Contexts / Occurrences / Attributes / Values: Dimensions, Sets, Individuals: (Mapping.of(...)). CSPO roles : Mapping roles according Contexts. Transforms: occurrences, attributes, values, aggregate, activate, align : Mapping<?>.

Example:

Kind(Context : Resource) : Context; Kind occurrences coming from previous Resource (navigation context).

Example Mapping:

(Context: SomeDimension, Occurrence: anIndividual Occurrence Resource Measure, Attributes, Values);

Set / Instance Mapping Contexts, Occurrences, Attributes, Values. Mapping Role types (Attributes / Values) for each Context / Occurrence types.

Render Mappings to / from Contexts / Occurrences / SPOs.

Augment Relationship Relations / Relation Relationships: Mapping, Statement, Resource transforms. Dimensional / Upper Augmentations. Reify / materialize to / from SPO Statements.

**Items:**

Implementation: RDF4J Core Spring / CDI configured.

MetaModel core RDFS schema / OWL SAIL. Layers.

Objects OGM / DCI schema SAIL.

Functional schema SAIL: Dataflow Augmentations, Metadata, Services.

Messages: SAIL Statements IO. Binder / Adapter reactive Message IO: Integrated sources.

Context: Layer / Instance Message shape activation: Quads, Object, Functional SAILs.

Hierarchies (Dataflow): Aggregation matches layer instances specializations. generalizations (layer instances) matches specialisations (layer instances). Aggregated Layers are recursively aggregated.

CDI Layers: SAILs / Rio IO Functional Java APIs Wrappers / Facades for Resources IO (connectors / services / protocols) for Reactive Dataflow Bus sources / processors / sinks. Reactive / Services / Events. SAILs / Rio IO.

CDI Layers: Domains / Contexts Services. Over CDI Layers Functional APIs Wrappers / Facades. Examples: Datasources (Drivers), OGM, OData, etc.

Naming: DIDs Services. [schema.org](http://schema.org), ont.io, [sameAs.org](http://sameas.org). Matching (mappings). WordNet, OpenNLP, Wikipedia. SAIL / Rio.

Reactive (Bus): Vert.x Dataflow: SCDF Pipelines.

Designer: OpenRefine. Protege Web. Editors / browsers (Forms / Flows).

Protocols (SAILs): SPARQL, JSON-LD, JDBC (OGM / OData: Driver DatabaseMetadata). Messages (Functional / Augmentations) Declarative Services (templates).

Clients:

Debug Console.

Declarative UI: ZK / XUL Templates (Content types / Components activation).

Services APIs: Declaratively stated services: templates / queries.

IDE. Runtime Deployment (Application).

Clients: OData / JDBC / WS-\* (CXF) / HATEOAS / etc.

JXTA. DHT. Kafka. Event sourcing. W3C DIDs. Smart Contracts. SoLiD. StratML.

URI: Jersey / CDI APIs. Persistence interface template methods.

(Statement, OldEmployees : Predicate URI / Activation, SalaryUpdate : Mapping URI / Aggregation, Percentage : Function URI / Alignment)

(Domain / Context matching signatures (async streams / topics), Employee : Aggregation / Predicate, Developer : Activation / Mapping / Attributes, ProgrammingLanguages / Alignment / Function / Values);

**To Do:**

* Eclipse.
* RDF4J.
* Core: Deploy / Spring / CDI (JEE). SAILs. Events APIs. Reactive. Vert.x / Kafka / others.
* Core: RDF Quads / Object Mappings / Functional SAILs. Persistence / Representations.
* Matching / Alignments Services: Resolver, Index, Naming, Registry. Lucene. FCA VSM. Events APIs. Reactive.
* Core Integration Services SAILs. Matching: Ontology / Domains / Flows Alignments:
* Apache MetaModel / JBoss Teiid / Services (REST / WS) / Data Sources. Events APIs. Reactive.
* JBoss KIE Services. Events APIs. Reactive.
* Core: Designer. Protege, OpenRefine. Jupyter Notebooks.
* Core: Business Services. OData / JDBC / JCA / OGM / JAF, REST HATEOAS / HAL, WS-\*, DCI / Qi4j. SAILs / Endpoints (Spring / CDI APIs).
* Core: Business Application Backend (Console / Apache Isis).
* Core: Business: Tryton Backend.

**Ontology Matching:**

The typical use case I have in mind is the declaration of equivalence between a flat statement and a property chain, as in the two patterns below:

@prefix : <http://example.org/> .

@prefix ex: <http://ontology.org/example> .

ex:book ex:hasauthor "John".

:book :hasbeencreated :creation\_event .

:creation\_event :carried\_out :person .

:person :is\_identified\_by :appellation .

:appellation rdfs:label "John" .

another important type of equivalence, but slightly different, which I would like to declare is the one between these two patterns:

@prefix : <http://example.org/> .

@prefix ex: <http://ontology.org/example> .

ex:Architect rdfs:label "John" .

:Person :classifiedAs <http://vocab.getty.edu/aat/300024987> ;

rdfs:label "John".

<http://vocab.getty.edu/aat/300024987> a gvp:Concept ;

rdfs:label "architects"@en .

The first one declare an instance of the class Artist according to an ontology(x), the second classify as artist, using a controlled vocabulary term, an instance of a person declared using the ontology (y).

Do you know how can I express such alignments?

I heard about EDOAL, but sincerely I did not fully grasped how to actually use it.

@prefix : <http://example.org/> .

@prefix ex: <http://ontology.org/example> .

ex:book ex:hasauthor "John".

:book :hasbeencreated :creation\_event .

:creation\_event :carried\_out :person .

:person :is\_identified\_by :appellation .

:appellation rdfs:label "John" .

In the scenario above, what is the relation between ex:book and :book, and can you define some rule on how to create one from the other? Is it "same-local-name-but-different-namespace"?

If ex:book and :book are identical then a SHACL sh:equals constraint can be used

ex:BookShape

a sh:NodeShape ;

sh:property [

sh:path ( :hasbeencreated :carried\_out :is\_identified\_by rdfs:label ) ;

sh:equals ex:hasauthor ;

] .

<https://www.w3.org/TR/shacl/#EqualsConstraintComponent>

Note that SHACL includes a syntax for SPARQL-like property path expressions, and the value of sh:path above is a property chain (represented as a simple rdf:List). Other types of paths are supported too: <https://www.w3.org/TR/shacl/#property-paths>

@prefix : <http://example.org/> .

@prefix ex: <http://ontology.org/example> .

ex:Architect rdfs:label "John" .

:Person :classifiedAs <http://vocab.getty.edu/aat/300024987> ;

rdfs:label "John".

<http://vocab.getty.edu/aat/300024987> a gvp:Concept ;

rdfs:label "architects"@en .

I am also unclear whether you want to use alignments to validate constraints (e.g. "does pattern 2 exist for pattern 1"), or to construct/infer one pattern out of the other.

The complexity of the scenario above indicates that you may want to use SPARQL, because then you can more easily look up values through matches, e.g. to match "architects"@en to ex:Architect using some look-up table, and because SPARQL gives you a maximum of expressiveness.

- If you want to validate constraints, you could use SHACL-SPARQL constraints: <https://www.w3.org/TR/shacl/#sparql-constraints>

- If you want to construct target triples, you could use SHACL-AF rules: <https://w3c.github.io/shacl/shacl-af/#rules>

You can write the alignment rules in RIF, then translate them to SPARQL

and use as needed, either as part of a runtime query, or to materialize

your own "equivalence" triples (using SPARQL INSERT or CONSTRUCT).

Rules written in RIF are easier to analyze and document, for example by

expressing them in the RIF XML notation and using XSLT.

I used this approach to align and validate part master and product

structure information in a large dataset derived from different PLM

systems.

What correspondences do you want to express exactly?

From your first example, I understand that you want to express that the

property:

ex:hasauthor

from the first ontology corresponds to the property chain:

:hasbeencreated o :carried\_out o :is\_identified\_by o rdfs:label

in the second ontology.

From your second example, I understand that you want that the class of

ex:Architect in the first ontology correspond to the class of people

that are classified as http://vocab.getty.edu/aat/300024987 .

I wrote these correspondences in an alignment file at:

https://www.emse.fr/~zimmermann/edoal-example.xml

The alignment format from Inria's alignment API is meant to represent

correspondences in a way independent from how the correspondence may be

used. There are different ways of interpreting and using a

correspondence in an alignment:

1. as an ontological axiom

2. as data transformations

3. as schema constraints

4. as "bridge rules" between descriptions of different contexts

Additionally, ontology alignment correspondences may have a "measure"

assigned to them that can be interpreted as a degree of confidence, or

as a fuzzy value, or as a probability, or something else. They also have

additional metadata that makes it clear that they are relating something

from an ontology to something from another ontology. In comparison, a

logical axiom, even if it uses URIs from different namespaces, does not

make this clear.

So, depending on your use case, you may want to use Holger's suggestion

(SHACL) or Paul's (RIF + SPARQL), or something else, but you may also

postpone the decision for later (or leave it to someone else) and just

write an EDOAL alignment like I did. The alignment file can also serve

other purposes, such as alignment evaluation, composition, and enrichment.

**HyTime:**

HyTime (use cases / implementations): XML, XPath, XSL, XLink, XPointer, XQuery use cases. Dimensional Addressing, Rendering (Layers) of Resources representations in contexts. Layers object model: domains to user / services gestures. Declarative templates. Objects DCI / MVC activation (JAF / CDI).

**Scheme:**

I am pleased to announce that we reached the last call for comments on

SRFI-168.

That is, a generic approach to triple and quad store. It support tuples of n-items.

While it might seem overkill to support tuples of n-items, it was somewhat

easy to implement (once you know the math: <https://math.stackexchange.com/q/3146568/23663>). Also, I need such a database

abstraction to implement something (that I think) will be suitable to version

tuples in a direct-acyclic graph (ala git).

This work was inspired from:

"Collaborative Open Data versioning: a pragmatic approach using Linked Data"

Federico Morando, Raimondo Iemma, Simone Basso et Lorenzo Canova

<https://iris.polito.it/handle/11583/2617308>

Hopefully, it will allow to build some kind of portable and scalable wikidata & wikibase.

The very last version of the specification can be read at:

<http://htmlpreview.github.io/?https://github.com/scheme-live/srfi-168/blob/master/srfi-168.html>

A few hints:

- It is supposed to be supported by an ordered key-value (wiredtiger, foundationdb, Oracle BerkeleyDB, TiKV, leveldb, rocksdb...), see SRFI-167 <http://htmlpreview.github.io/?https://github.com/scheme-live/srfi-167/blob/master/srfi-167.html>.

- It can support (only!) scheme base data types but not complex ie. bigint, float, string, symbol, bytevector and vector and list of those preceding. This is implementation dependant but the implementations so far only support those.

- The query language is based on stream-processing. All queries are at least O(n) where n is the length of the seed generator (nstore-from).

- The specification is written in a way that will allow support geospatial indexing

directly in the nstore and outside the nstore but inside the same database so that ACID semantic still holds. More generally, the nstore abstraction can live along side other abstractions in the same okvs database, which allows to implement specific indexing scheme to support geospatial indices or better pagination.

It is still time to make feedback and I hope you do :)

Amirouche ~ amz3

Hello Sebastian,

Glad to read about someone interested in this project.

Le lun. 10 juin 2019 à 17:25, Sebastian Samaruga <ssamarug@gmail.com> a écrit :

Hi,

Sorry for my ignorance but I don't understand "the math".

Me neither, I just explained my problem several times on mathoverflow and at some point, someone came up with an explanation and some else with an algorithm.

Could you please better explain for me what are "prefix permutations"

Good question! The definition is in the stackoverflow question. That is: A is "prefix-permutation" of B, if there is a permutation of A that is prefix of B.

and how they are useful for building indices for SPARQL like queries?

This is useful to me because I rely on an ordered key-value database that is bytes to bytes hashmap where keys are "sorted". Using that database

it is very easy to query for all keys by prefix. Now, when you consider SPARQL queries for a triplestore the WHERE clauses look like the following

(subject, predicate, value)

In my implementation, I consider that ONE or more can be a variable. We can enumerate

those patterns:

(?subject, predicate, object)

(?subject, ?predicate, object)

(?subject, ?predicate, ?object)

(?subject, predicate, ?object)

(subject, ?predicate, ?object)

(subject, predicate, ?object)

(subject, ?predicate, object)

You can recognize that the combinations of (subject, predicate, object) describe all

the patterns, here are those combinations:

('subject',)

('predicate',)

('object',)

('subject', 'predicate')

('subject', 'object')

('predicate', 'object')

So, to be able to bind ANY pattern (in one hop). For all combinations, there must be an index

that can COMPLETE the given combination to form a complete tuple. One can recognize that

the order in which the items of the combination appears in the index doesn't matter. To take the

previous example, both of those indices:

(predicate, object, subject)

And

(object, predicate, subject)

Allow to complete the following pattern:

(?subject, predicate, object)

So the algorithm minimize the set of indices given that prefix-permutation property.

There must be a better explanation.

I've been done some permutations / combinations (rotations) in a model layer of an ongoing project, but still being a newbie and doing much by intuition.

If you dare to skim a very fuzzy and (incoherent?) early drafts, check "Index.docx" at:

https://github.com/snxama/scrapbook

I just skimmed over it. It very fuzzy. I think it would help if you had a table of content and write down the introduction and the conclusion of your work beforehand.

I don't know if this is feedback for what you are doing. I'm trying to build core ontologies for semantic / object mappings for declarative modelling of "business domains" inferred from aggregating and blending distributed sources of data.

aggregating different sources of data is difficult. I will not forbid you from trying.

As I'm in a stage of choosing a platform for the development of some kind of prototype, I'll be choosing an n-store implementation.

I would be very glad if you implemented my nstore in Java. Wiredtiger has bindings in Java. There might other bindings or ordered key-value store available in Java. FoundationDB is available in Java.

Maybe, I think, yours seems very "flexible" although I haven't reviewing very much other alternatives yet.

That work started as a learning project. Like I said in the original mail, I stumble upon an interesting problem (how to version triples) and thought that it would be fun to solve.

Now, it happens that for my (want-to-be) natural language processing (want-to-be) research, I will use the nstore.

Feel free to ask more question. Also I am following you on github.

Sure!

You can watch this repository, eventually, that is where I will put everything https://github.com/awesome-data-distribution/datae

Le lun. 10 juin 2019 à 20:43, Sebastian Samaruga <ssamarug@gmail.com> a écrit :

Hello,

Thanks a lot for your response. I'd like to hear more about your project. For Java, there is Clojure, a LISP dialect that runs natively and integrates seamlessly with all Java ecosystem, frameworks and other VM languages.

Maybe I'm so dumb but, could you define the predicate isPrefixOf(statement1, statement2).

"abc" is prefix of "abcdef".

What makes a tuple being prefix of another?

It is not a tuple that is prefix of another. It is the combination (or a permutation of it) that is prefix of the tuple representing the index.

For instance, the following combination:

(subject, predicate)

is prefix of:

(subject, predicate, object)

(object) is prefix of:

(object, subject, predicate)

Also (subject, object) is prefix of

(subject, object, predicate)

and (subject, object) permutation-prefix of:

(object, subject, predicate)

because there is a permutation that is (object,subject) that is prefix of it.

Hope it helps!

I created a repository to make it easier to test a real implementation

of that specification: <https://github.com/scheme-live/srfi-167-and-168-tutorial#setup>

On preliminary benchmarks that code is faster than blazegraph!

This is not perfect (one of the cloned repository is 1G big) but it should get you going.

Also it does require a linux box but doesn't require system installation.

I made progress regarding the functional store aka. fstore ie. a versioned generic

tuple store (which is the reason I needed the nstore and eventually discovered it

can be useful in other contexts).

It is bundled in the above repository, it is not documented [1] but there is a beginning of test

suite at:

https://git.sr.ht/~amz3/chez-scheme-arew/tree/master/src/arew/data/base/fstore-test.scm

[1] well, the API of fstore is the same as the API nstore, except you have to specify a branch

against which the query must happen.

Le sam. 8 juin 2019 à 13:51, Amirouche Boubekki <amirouche.boubekki@gmail.com> a écrit :

I am pleased to announce that we reached the last call for comments on

SRFI-168.

That is, a generic approach to triple and quad store. It support tuples of n-items.

While it might seem overkill to support tuples of n-items, it was somewhat

easy to implement (once you know the math). Also, I need such a database

abstraction to implement something (that I think) will be suitable to version

tuples in a direct-acyclic graph (ala git).

I got a question off-list about what permutation-prefix means and where it is

useful.

There is a sister question on stackoverflow that try to explain in terms of programming what problem I am trying to solve: <https://stackoverflow.com/q/55143485/140837>

I will work on a document that will better explain how it is used and why

it is useful.

I am very pleased to announce the immediate availability of nomunofu.

nomunofu is database server written in GNU Guile that is powered by WiredTiger ordered key-value store.

It allows to store and query triples. The goal is to make it much easier, definitely faster to query as big as possible tuples of three items. To achieve that goal, the server part of the database is made very simple and it only knows how to do pattern matching. Also, it is possible to swap the storage engine to something that is horizontally scalable and resilient.

The client must be smarter, and do as they please to full-fill user requests. Today release only include a minimal Python client. In the future, I plan to extend the Python client to fully support SPARQL 1.1.

Preliminary tests over 100 000 and 1 000 000 triples are good looking. Next step is to reach 1 billion triples and eventually 9 billions wikidata triples.

You can get the code with the following command:

git clone https://github.com/amirouche/nomunofu

After the installation of GNU Guix [0], you can do:

make init && gunzip test.nt.gz && make index && make web

And in another terminal:

make query

Thanks to Guix, portable binaries for amd64 Ubuntu 18.04 will be made available in a few weeks, along with this, a docker image will be built. The binary release will include wikidata pre-loaded.

[0] https://guix.gnu.org/download/

Here is an example ipython session:

$ ipython

Python 3.7.3 (default, Oct 7 2019, 12:56:13)

Type 'copyright', 'credits' or 'license' for more information

IPython 7.10.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from nomunofu import Nomunofu

In [2]: from nomunofu import var

In [3]: nomunofu = Nomunofu('http://localhost:8080');

In [4]: nomunofu.query((var('uid'), "http://www.w3.org/2000/01/rdf-schema#label", "Belgium"))

Out[4]: [{'uid': 'http://www.wikidata.org/entity/Q31'}]

In [5]: nomunofu.query((var('uid'), "http://www.w3.org/2000/01/rdf-schema#label", "Belgium"), (var('about'), "http://

...: schema.org/about", var('uid')))

Out[5]:

[{'uid': 'http://www.wikidata.org/entity/Q31',

'about': 'https://www.wikidata.org/wiki/Special:EntityData/Q31'},

{'uid': 'http://www.wikidata.org/entity/Q31',

'about': 'https://it.wikivoyage.org/wiki/Belgio'},

{'uid': 'http://www.wikidata.org/entity/Q31',

'about': 'https://an.wikipedia.org/wiki/Belchica'},

{'uid': 'http://www.wikidata.org/entity/Q31',

'about': 'https://sl.wikipedia.org/wiki/Belgija'},

{'uid': 'http://www.wikidata.org/entity/Q31',

'about': 'https://pfl.wikipedia.org/wiki/Belgien'},

{'uid': 'http://www.wikidata.org/entity/Q31',

'about': 'https://crh.wikipedia.org/wiki/Bel%C3%A7ika'},

{'uid': 'http://www.wikidata.org/entity/Q31',

'about': 'https://fiu-vro.wikipedia.org/wiki/Belgi%C3%A4'},

{'uid': 'http://www.wikidata.org/entity/Q31',

'about': 'https://fr.wikipedia.org/wiki/Belgique'}

...

Cheers,

Amirouche ~ zig ~ <https://hyper.dev>

Le dim. 8 déc. 2019 à 18:52, Amirouche Boubekki

<amirouche.boubekki@gmail.com> a écrit :

>

> I am very pleased to announce the immediate availability of nomunofu.

>

> nomunofu is database server written in GNU Guile that is powered by WiredTiger ordered key-value store.

>

> It allows to store and query triples. The goal is to make it much easier, definitely faster to query as big as possible tuples of three items. To achieve that goal, the server part of the database is made very simple and it only knows how to do pattern matching. Also, it is possible to swap the storage engine to something that is horizontally scalable and resilient.

>

I pushed portable binaries built with gnu guix for amd64 with a small

database file. You can download it with the following command:

$ wget https://hyper.dev/nomunofu-v0.1.3.tar.gz

The uncompressed directory is 7GB.

Once you have downloaded the tarball, you can do the following cli

dance to run the database server:

$ tar xf nomunofu-v0.1.3.tar.gz && cd nomunofu && ./nomunofu serve 8080

The database will be available on port 8080. Then you can use the

python client to do queries.

Here is example run on the current dataset, that queries for instance

of (P31) government (Q3624078):

In [1]: from nomunofu import Nomunofu

In [2]: from nomunofu import var

In [3]: nomunofu = Nomunofu('http://localhost:8080');

In [4]: nomunofu.query((var('uid'),

'http://www.wikidata.org/prop/direct/P31',

'http://www.wikidata.org/entity/Q3624078'), (var('uid'),

'http://www.w3.org/2000/01/rdf-schema#label', var('label')))

Out[4]:

[{'uid': 'http://www.wikidata.org/entity/Q31', 'label': 'Belgium'},

{'uid': 'http://www.wikidata.org/entity/Q183', 'label': 'Germany'},

{'uid': 'http://www.wikidata.org/entity/Q148', 'label': 'China'},

{'uid': 'http://www.wikidata.org/entity/Q148',

'label': "People's Republic of China"},

{'uid': 'http://www.wikidata.org/entity/Q801', 'label': 'Israel'},

{'uid': 'http://www.wikidata.org/entity/Q45', 'label': 'Portugal'},

{'uid': 'http://www.wikidata.org/entity/Q155', 'label': 'Brazil'},

{'uid': 'http://www.wikidata.org/entity/Q916', 'label': 'Angola'},

{'uid': 'http://www.wikidata.org/entity/Q233', 'label': 'Malta'},

{'uid': 'http://www.wikidata.org/entity/Q878',

'label': 'United Arab Emirates'},

{'uid': 'http://www.wikidata.org/entity/Q686', 'label': 'Vanuatu'},

{'uid': 'http://www.wikidata.org/entity/Q869', 'label': 'Thailand'},

{'uid': 'http://www.wikidata.org/entity/Q863', 'label': 'Tajikistan'},

{'uid': 'http://www.wikidata.org/entity/Q1049', 'label': 'Sudan'},

{'uid': 'http://www.wikidata.org/entity/Q1044', 'label': 'Sierra Leone'},

{'uid': 'http://www.wikidata.org/entity/Q912', 'label': 'Mali'},

{'uid': 'http://www.wikidata.org/entity/Q819', 'label': 'Laos'},

{'uid': 'http://www.wikidata.org/entity/Q298', 'label': 'Chile'},

{'uid': 'http://www.wikidata.org/entity/Q398', 'label': 'Bahrain'},

{'uid': 'http://www.wikidata.org/entity/Q12560', 'label': 'Ottoman Empire'}]

As of right now there is less than 10 000 000 triples that were

imported. Blank nodes are included, but only english labels are

imported.

You can find the source code at:

https://github.com/amirouche/nomunofu

I hope you have a good day!

I am pleased to share with you the v0.1.4 binary release. It contains

the following improvements:

- The REST API takes JSON as input, which will make it easier to

create clients in other programming languages;

- The REST API takes limit and offset as query string. The maximum

limit is 1000;

- There is better error handling, the server will return a HTTP status

code 400 if it detects an error;

- Add aggregation queries `sum`, `count` and `average`, see the Python

client (nomunofu.py) to know how to properly format the query;

- Python client method `Nomunofu.query(\*patterns, limit=None,

offset=None)` returns a generator.

Also, the harmless warnings are silenced. The database files are

compatible with the previous release.

This release comes with full wikidata lexemes triples.

You can download the amd64 portable binary release plus database files

with the following command:

wget http://hyper.dev/nomunofu-v0.1.4.tar.bz2

The directory is 11G uncompressed.

Grab the source code with the following command:

git clone https://github.com/amirouche/nomunofu

Here is an example Python query that returns at most 5 adverbs:

In [10]: for item in nomunofu.query(

...: (var('uid'), wikibase('lexicalCategory'),

'http://www.wikidata.org/entity/Q380057'),

...: (var('uid'), rdfschema('label'), var('label')),

...: limit=5):

...: print(item)

...:

{'uid': 'http://www.wikidata.org/entity/L3244', 'label': 'always'}

{'uid': 'http://www.wikidata.org/entity/L4124', 'label': 'here'}

{'uid': 'http://www.wikidata.org/entity/L4326', 'label': 'often'}

{'uid': 'http://www.wikidata.org/entity/L5201', 'label': 'too'}

{'uid': 'http://www.wikidata.org/entity/L5321', 'label': 'yet'}

Cheers,

Hello all ;-)

I ported the code to Chez Scheme to do an apple-to-apple comparison

between GNU Guile and Chez and took the time to launch a few queries

against Virtuoso available in Ubuntu 18.04 (LTS).

Spoiler: the new code is always faster.

The hard disk is SATA, and the CPU is dubbed: Intel(R) Xeon(R) CPU

E3-1220 V2 @ 3.10GHz

I imported latest-lexeme.nt (6GB) using guile-nomunofu, chez-nomunofu

and Virtuoso:

- Chez takes 40 minutes to import 6GB

- Chez is 3 to 5 times faster than Guile

- Chez is 11% faster than Virtuoso

Regarding query time, Chez is still faster than Virtuoso with or

without cache. The query I am testing is the following:

SELECT ?s ?p ?o

FROM <http://fu>

WHERE {

?s <http://purl.org/dc/terms/language> <http://www.wikidata.org/entity/Q150> .

?s <http://wikiba.se/ontology#lexicalCategory>

<http://www.wikidata.org/entity/Q1084> .

?s <http://www.w3.org/2000/01/rdf-schema#label> ?o

};

Virtuoso first query takes: 1295 msec.

The second query takes: 331 msec.

Then it stabilize around: 200 msec.

chez nomunofu takes around 200ms without cache.

There is still an optimization I can do to speed up nomunofu a little.

Happy hacking!

<https://github.com/amirouche/copernic>

<http://copernic.space/>

Scheme tuple store: Rio / SAIL (Form / Flows Cons Zipper Shapes).

**Zippers:**

Zippers: car / cdr on tree / list structures with predicates / iteration / recursion (reference model, contexts / occurrences). Shape Monads. E.g.: Uncle (reified relation predicate and reified Relation w./ roles / attributes). Dynamic Shape Monad on Kinds. Aggregation, Activation, Alignments.

Example: Marriage (TBD).

Predicates:

:aHusband :marriedWith :aWife

:marriedWith rdfs:domain :Male

:marriedWith rdfs:range :Female

Relationship:

(aMarriage : Relation, anStatement : marriageStatement, aKind : husbandRole, aResource : aHusband);

(aMarriage : Relation, anStatement : marriageStatement, aKind : wifeRole, aResource : aWife);

(Marriage : Relationship, Marriages : Relation, anStatement : marriagesStatements, aKind : marriageRole);

Predicates / Relationships, Relationships / Predicates entailment. Dimensional: inference / relation types / restrictions.

Encode order / hierarchies / relations (parent / child, prev / next, etc.) / iterations / conditionals / jumps.

Dimensional Domain: dimensions, units, measures, values. Comparisons, relations. State. Events (marriage example). Verbs (action, passion, state). Order (data / schema / behavior).

**Proposal for representing Aggregate Statistical Data (Guha):**

This document can be accessed here:

<https://docs.google.com/document/d/139jXakeQk4ChwCkGjqq5wJfCPMDnwIV94oCH-JzJrhM/edit?usp=sharing>

Look forward to feedback.

Guha

Representing aggregate statistics

Examples of aggregate statistical reports include those from Census Organizations (e.g., American Community Survey), Health Organizations (e.g., CDC Wonder) and many others. This is a schema, currently in use on DataCommons.org for representing facts stated in these reports. This document describes certain general mechanisms for representing statistical populations and associated observations. This document will be followed later by a companion proposal suggesting some basic common vocabulary useful for representing the kind of data released by the US Census, CDC, etc.

Our interest is not in describing a data set or mapping columns in csv files, but in representing the actual data itself. Other efforts have focused on characterizing data cubes in terms of dimensions, etc. While we draw upon their work, our goals are different.

Examples of the kind of statistics we would like to represent include:

1. In 2016, there were 1213 people in East Podunk, California, who were male, married, with a median age of 22.

2. In 2017, there were 20 deaths in Falooda County where the cause of death was XYZ

We will refer to ‘number of people who are male, hispanic’, ‘number of deaths where cause of death was XYZ’, etc. as variables. Since the number of possible variables increases combinatorially, clearly, we can’t have a properties for each variable (or worse, property for each variable x years). We need a way of compositional way of constructing variable references. We use the concept of a StatisticalPopulation to do this construction.

A StatisticalPopulation is a set of instances of a certain given type that satisfy some set of constraints. The property populationType is used specify the type. Any property that can be used on instances of that type can appear on the statistical population. An instance of StatisticalPopulation whose populationType is C1, which has the properties p1, p2, … with values v1, v2, … corresponds to the set of objects of type C1 what have the property p1 with value v1, property p2 with value v2, etc. The properties numConstraints and constrainingProperties are used to specify which of the populations properties are used to specify the population. In the two examples above:

Node: SP1

type: StatisticalPopulation

populationType: Person

location: EastPodunkCalifornia

gender: Male

maritalStatus: Married

numConstraints: 3

constrainingProperties: location, gender, race

Node: SP2

type: StatisticalPopulation

populationType: MortalityEvent

location: FaloodaCounty

causeOfDeath: XYZ

numConstraints: 2

constrainingProperties: location, causeOfDeath

SP1 is an abstract set in the sense that it does not correspond to a particular set of people who satisfy that constraint at a certain point in time, but rather, to an abstract specification, about which we can make observations that are grounded at a particular point in time. We now turn our attention to the representation of these observations.

Instances of the class Observation are used to specify observations about an entity (which may or may not be an instance of a StatisticalPopulation), at a particular time. The principal properties of an Observation are observedNode, measuredProperty, measuredValue (or median, etc.) and observationDate (measuredProperty can, but need not always, be w3c rdf data cube "measure properties", as in lifeExpectancy example here: https://www.w3.org/TR/vocab-data-cube/#dsd-example.) In the two examples above:

Node: Obs1

type: Observation

observedNode: SP1

measuredProperty: age

median: “23 years”

observationDate: “2016”

Node: Obs2

type: Observation

observedNode: SP1

measuredProperty: count

measuredValue: 1213

observationDate: “2016”

Node: Obs3

type: Observation

observedNode: SP2

measuredProperty: count

measuredValue: 20

observationDate: “2017”

Observations can also have properties related to the measurement technique, margin of error, etc. To elaborate on Obs2 above, we can have:

Node: Obs2

type: Observation

observedNode: SP1

measuredProperty: count

measuredValue: 1213

observationDate: “2016”

marginOfError: 22

measurementMethod: CensusACS5yrSurvey

Notes:

1. Care needs to be exercised when querying StatisticalPopulations, to make sure that the query specifies all the constraining properties.

2. We do not yet have a way of using properties which are named in the opposite direction e.g. we handle "alumniOf" (relating a person to an org), but if the only existing property was "alumni" (relating an org to a person).

**HL7**

HL7 v3 can not be fully implemented with the current state of Object Databases.

I'm interested in HL7 and in particular FHIR.

FHIR has an RDF/JSONLD representation, and the schema is represented as SHEX (amongst several other implementations, including JSON/JSON schema, XML/RelaxNG etc).

I think this is a very interesting piece of work and I'm interested in collaborations.

I am participating in the European H2020 project named GATEKEEPER which is focused on smart personal healthcare. HL7 is a member of the project. We’re looking at combining the Web of Things, FHIR, semantic technologies and more in a variety of pilots. FHIR is essential a means to use RESTful interfaces to access healthcare related data in a variety of formats, e.g. JSON-LD. There is a body of work on healthcare related ontologies. Here is a pointer to FHIR by the UK’s NHS, and the overview from FHIR itself:

<https://digital.nhs.uk/services/fhir-apis>

<https://www.hl7.org/fhir/overview.html>

A major challenge is how to integrate heterogeneous data sources whilst avoiding a loss of valuable information due to differences in vocabularies, data quality, context and so forth.

**StratML:**

Can AI be used on StratML utilizing ontologies to create customizable dashboards for project management and collaboration for large networks of collaborating people from different fields of work?

Question is of importance in eGovernment, eGovernance and achieving sustainable development goals.

XML constitutes a structure for language, and certain kinds of AI can be built

using structured language expressions.

According to Wikipedia, intelligence involves perceiving the environment and acting to maximize the chance of achieving goals. https://en.wikipedia.org/wiki/Artificial\_intelligence The article also cites Tesler's Theorem: "AI is whatever hasn't been done yet." See also https://en.wikipedia.org/wiki/AI\_effect

While it may be possible for intelligent agents to decipher goals from unstructured text, it seems likely they might be able to more effectively help us achieve our goals if we make them explicit in terms of near-term objectives and performance indicators. Since that has not yet been done on a worldwide scale, why should it not begin with us?

There are more than 4,000 plans in the StratML collection that can be used for demonstration purposes, including the SDGs: <https://stratml.us/drybridge/index.htm#UNSDG>

As far as I am aware, however, progress against the SDGs is not being reported in an open, standard, machine-readable format. See, for example, the HTML, CSV & PDF at https://unstats.un.org/sdgs/report/2019/ The presentation of the data is actually pretty good. See, for example, https://unstats.un.org/sdgs/report/2019/goal-17/

The problem is that few people know about and take the time to view it, much less to do anything about it. While making the data available in open, standard, machine-readable format would not solve that problem, it would make it easier for value-added intermediaries to engage stakeholders in ways that are more accessible and meaningful to them, e.g., in their own personal and organizational/corporate performance plans. From my perspective, failing to do so is an example of artificial ignorance. https://www.linkedin.com/pulse/artificial-ignorance-owen-ambur/

Here's an article addressing the role of AI in achieving the SDGs: https://www.nature.com/articles/s41467-019-14108-y It is relatively long and unstructured and I haven't taken the time to read all of it. However, here some key point:

We therefore recommend that AI applications that target SDGs are open and explicit about guiding ethical principles, also by indicating explicitly how they align with the existing guidelines.

See StratML tool, app, and service requirements Objective 8.3: Values Alignment & Goal 9: Values Validation. Those requirements are prime candidates for the application of AI.

On the other hand, the lack of interpretability of AI, which is currently one of the challenges of AI research, adds an additional complication to the enforcement of such regulatory actions ...

AI developers and agents should be expected, if not required, to document their objectives and performance indicators in an open, standard format that is both human- and machine-readable.

... associations such as the Future of Life Institute are reviewing and collecting policy actions and shared principles around the world to monitor progress towards sustainable-development-friendly AI ...

Perhaps we should schedule a televideo conference for them to brief us on their AI activities and explore prospects for collaboration. <https://stratml.us/carmel/iso/FLIwStyle.xml>

A global and science-driven debate to develop shared principles and legislation among nations and cultures is necessary to shape a future in which AI positively contributes to the achievement of all the SDGs.

Whenever I see calls for legislation and regulation, I view it both as a cop out as well as an attempt to dictate to others that which we should take upon ourselves to do, in partnership with those who share our values and objectives. Perhaps those who view it as the best way forward should either study the China model ... or just stand idly by and wait for them to impose it on all of us. For an alternative model, see https://www.linkedin.com/pulse/privately-well-practiced-public-policymaking-owen-ambur/

Here's a shorter article on AI & the SDGs: https://www.undp.org/content/undp/en/home/blog/2019/Using\_AI\_to\_help\_achieve\_Sustainable\_Development\_Goals.html It suggests, "To improve data accessibility, for example, collectors and generators of data, whether governments or companies, will need to grant greater access to NGOs and others seeking to use the data for public service." However, it shows no awareness of the importance of open, machine-readable data standards -- perhaps because usage of such standards would reduce the need for consultants ... or, rather, it would require them to add higher-level values than massaging amorphous, aimless data.

With respect to dashboards, see https://gcn.com/articles/2017/09/12/yet-another-dashboard.aspx & https://gcn.com/articles/2017/05/10/machine-readable-data.aspx

I look forward to learning what we might be able to do together along these lines.

**Monads:**

Type constructor (wrapper / type).

Unit wrapper / value instance.

Bind / map / flatMap. Instance argument, transform (static / functor). Instance method (applications).

Functor: A category consists of a collection of nodes (objects) and morphisms (functions). An object could be numbers, strings, urls, customers, or any other way you wish to organize like-things. (X, Y, and Z in the graphic are the objects.).

A map is a function to convert something from one object to another. (f, g, and fog are the maps). Google tip: A map between objects is called a Morphism.

So an object could be simple like a Number or a String. An object could also be more abstract like a Username, A User API URL, User API HTTP Request, User API Response, User API Response JSON. Then we can create maps or morphisms between each object to get the data we want.

Examples of morphisms: Username -> User API UrlUser API Url -> User API HTTP RequestUser API HTTP Request -> User API ResponseUser API Response -> User API Response JSON

Google tip: Function Composition is a way to combining multiple map or morphisms to create new maps. Using Function Composition we could create a map from Username directly to User API Response JSON.

Now that we understand what it means to be Mappable, we can finally understand what a Functor is.

A Functor is something that is Mappable or something that can be mapped between objects in a Category.

An Array is Mappable, so it is a Functor. In this example I am taking an Array of Numbers and morphing it into an Array of Strings.

Note: One of the properties of a Functor is that they always stay that same type of Functor. You can morph an Array containing Strings to Numbers or any other object, but the map will ensure that it will always be an Array. You cannot map an Array of Number to just a Number.

We can extend this Mappable usefulness to other objects too! Let's take this simple example of a Thing.

If we wanted to make Thing mappable in the same way that Array is mappable, all we have to do is give it a map(morphism) function. And that is a Functor! It really is just that simple. Google tip: The "Thing" Functor we created is known as Identity.

Monad:

Sometimes functions return a value already wrapped. This could be inconvenient to use with a Functor because it will re-wrap the Functor in another Functor.

This is where flatMap comes in handy. It's similar to map, except the morphism is also expected to perform the work of wrapping the value.

Summary:

A Functor is something that is Mappable or something that can be mapped between objects in a Category.

A Monad is similar to a Functor, but is Flat Mappable between Categories.

flatMap is similar to map, but yields control of the wrapping of the return type to the mapping function.

Monads:

Monads are a way to compose type lifting functions: g: a => M(b), f: b => M(c). To accomplish this, monads must flatten M(b) to b before applying f(). In other words, functors are things you can map over. Monads are things you can flatMap over:

A monad is a way of composing functions that require context in addition to the return value, such as computation, branching, or I/O. Monads type lift, flatten and map so that the types line up for lifting functions a => M(b), making them composable. It's a mapping from some type a to some type b along with some computational context, hidden in the implementation details of lift, flatten, and map:

Functions map: a => b which lets you compose functions of type a => b

Functors map with context: Functor(a) => Functor(b), which lets you compose functions F(a) => F(b)

Monads flatten and map with context: Monad(Monad(a)) => Monad(b), which lets you compose lifting functions a => F(b)

Map means, “apply a function to an a and return a b". Given some input, return some output.

Context is the computational detail of the monad’s composition (including lift, flatten, and map). The Functor/Monad API and its workings supply the context which allows you to compose the monad with the rest of the application.

The point of functors and monads is to abstract that context away so we don’t have to worry about it while we’re composing things. Mapping inside the context means that you apply a function from a => b to the value inside the context, and return a new value b wrapped inside the same kind of context.

Observables on the left? Observables on the right:

Observable(a) => Observable(b).

Arrays on the left side? Arrays on the right side:

Array(a) => Array(b).

Type lift means to lift a type into a context, blessing the value with an API that you can use to compute from that value, trigger contextual computations, etc… a => F(a) (Monads are a kind of functor).

Flatten means unwrap the value from the context. F(a) => a.

Dataflow, Reactive: Function composition creates function pipelines that your data flows through. You put some input in the first stage of the pipeline, and some data pops out of the last stage of the pipeline, transformed. But for that to work, each stage of the pipeline must be expecting the data type that the previous stage returns.

A monad is based on a simple symmetry A way to wrap a value into a context, and a way to unwrap the value from the context:

Lift/Unit: A type lift from some type into the monad context: a => M(a)

Flatten/Join: Unwrapping the type from the context: M(a) => a

And since monads are also functors, they can also map:

Map: Map with context preserved: M(a) -> M(b)

Combine flatten with map, and you get chain — function composition for lifting functions, aka Kleisli composition:

FlatMap/Chain Flatten + map: M(M(a)) => M(b)

Monads must satisfy three laws (axioms), collectively known as the monad laws:

Left identity: unit(x).chain(f) ==== f(x)

Right identity:[m.chain](about:blank)(unit) ==== m

Associativity:[m.chain](about:blank)(f).chain(g) ==== [m.chain](http://m.chain)(x => f(x).chain(g)

Monad: wrapper type. Metaclass.

Monad: wrapped type. Class.

Monad: wrapped value. Instance.

Monad: wrappers hierarchy context type instance. Occurrence (mapping results).

ISO:

About Relationship: many Relationship instances are the result of an Activity, e.g. Marrying – Marriage, Assembling – Assembly, Containing – Containment, Connecting – Connection, Employing – Employment, etc.

We model that by typing a Relationship with a (meta) ClassOfRelationshipWithSignature that is defined as a ClassOfRelationshipWithSignature is a ClassOfRelationship that may have a RoleAndDomain specified for each end. (where RoleAndDomain simply stands for ‘a Class in a Role’)

The instance of ClassOfActivity CONNECTING-A-TRAIN

:CONNECTING-A-TRAIN rdf:type dm:ClassOfActivity .

:CONNECTING-A-TRAIN :hasPartiipant1 RoleAndDomain1 .

:CONNECTING-A-TRAIN :hasPartiipant2 RoleAndDomain2 .

:RoleAndDomain1 rdfs:subClassOf rdl:LOCOMOTIVE .

:RoleAndDomain1 rdfs:subClassOf rdl:PULLER .

:RoleAndDomain2 rdfs:subClassOf rdl:TRAIN WAGON .

:RoleAndDomain2 rdfs:subClassOf rdl:PULLED .

The instance of ClassOfRelationshipWithSignature

:CONNECTION-OF-A-TRAIN rdf:type dm:ClassOfRelationshipWithSignature .

CONNECTION-OF-A-TRAIN :hasClassOfEnd1 RoleAndDomain1 .

:CONNECTION-OF-A-TRAIN :hasClassOfEnd2 RoleAndDomain2 .

The typed Relationship

:myRelationship rdf:type tpl:CONNECTION-OF-A-TRAIN ;

:myRelationship :hasPuller myLocomotive ;

:myRelationship :hasPulled myTrainWagon .

An instance of Relationship, typed with this metaclass CONNECTION-OF-TRAIN, can be linked to an instance of Activity, typed with ClassOfActivity CONNECTING-A-TRAIN, with an instance of above CauseOfEvent.

When I connect a train I cause the Event ‘train is connected’, which leads to a state that the locomotive and the trainwagon instances are connected, a fact that is recorded with an instance of ConnectionOfTrain relationship.

Data: product price / marital status.

Information: price variation / state change.

Knowledge: increase, decrease / marriage, divorce.

TBD.

**Notas:**

**Workflows (Domain Goals) framework:**

Semantic Distributed Backends StratML rendered into vitualized applications environment (collaborative dashboards, wizards, guided flows).

Workflow kind: Need, Good, Product. Roles. Inter-domain workflows.

Prompts: context wiki (domain learning roles, domain evaluations: skills).

Prompts: Resource matching domain skills answers flows: values / decisions.

Prompt: value / decision query. User / Service response.

Workflow tray: declarative API interface (workflow concepts). Guided assistant / learning.

App: questions with a purpose. Domains roles / skills filters.

Workflows / Prompts Model Relations / Services:

Domains: Data, Information, Knowledge.

Products / Goods / Needs.

Goals: Use Cases. Data, Contexts, Interactions.

Profiles.

Roles.

Organizations (ad hoc).

Workflows / Prompts Model Relations / Services:

(Domain, Flow : Flow, Role : Relationship, Item : Entity);

Domain<Flow[]>;

Flow: Use Case. Produced / available for Domains.

Role: Prompts. Produces / available in state of Flow. Item: product, prompts consumer / producer kind.

Item: Product / Good / Need (Goal). Produces / populated by Roles (Prompts).

(DomainService : Relation, Resource : Domain, Context : Flow, Role : Relationship);

Retrieves available workflows Roles for Flows in Domains (ActivationService).

**Meta Meta Model:**

DOM OntResource: Models Meta Meta Model. Reference Model. Runtime / dataflow APIs.

Parsing: AST Object URL monad layers hierarchy. Parser combinators. Monad Zippers (tree encoding). Order. Flows (streams). Parser combinators.

OntResource:

Reactive (producer / consumer) entity. Hierarchy templates determines signatures / dataflows (topics / queues).

Signatures: observer (topics): ObjectID events stream. URL events stream. Role events stream. Layers hierarchy events set (reified: Resource, Statement, Kind, Relation events). Filter. Template / abstract methods. Core OntResource API behaviors. Dataflow.

Signatures: observer (topics): ObjectID events stream. URL events stream. Role events stream. Layers hierarchy events set (reified: Resource, Statement, Kind, Relation events). Filter. Template / abstract methods. Core OntResource API behaviors. Dataflow.

OntResource abstract APIs:

Abstract: Meta Model objects occurrences (aggregated by Object URL and Roles):

OntResource::ObjectID;

OntResource::Occurrences<URL, Role> : OccurrenceID;

OntResource represents aligned Objects / Resources of which the same Object (ObjectID) occurs in different Roles and (possibly) diferent URLs. Matching and alignments by reifying layers into abstract resources / OntResources.

Meta Model(s) APIs (Augmentation, Dataflow, Dimensional) reflects reference model events reacting to and producing signature matching messages on Meta Model(s).

OntResource's Roles are the DOM (Dynamic Object Model) types of the Meta Model(s) layers Occurrences / Objects / Contexts / SPOs. FCA Lattice contexts occurrences metadata. DOM APIs (reifying hierarchy layers, abstract templates):

Role: occurrence / object in CSPO slots. Denotes resource types in positions in statements (i.e.: Kind in Relation). Role CSPO is object / occurrence in statement occurrence position, Role type (i.e.: Kind, Relation) stated as Role instances in Meta Models with corresponding Kinds for its complimentary CSPO resources.

(Instance, Class, Metaclass, Occurrence);

(Context, Occurrence, Attribute, Value);

Meta Model(s). Augmentation APIs hierarchy:

Object<URL> : OntResource;

Abstract. OntResource aligned URL / template method design pattern.

Object<URL>;

Abstract. ObjectID.

Occurrence : Object, Object<URL> : OntResource;

Abstract ObjectID, OccurrenceID. Reify aggregated hierarchies.

Role : Occurrence, Occurrence, Object<URL>;

(Resource : Role, Role, Occurrence, Object<URL>);

(Kind : Resource, Resource, Role, Occurrence);

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

(Relation : Statement, Statement, Kind, Resource);

Class hierarchy APIs:

(Instance, Class, Metaclass, Occurrence);

(Context, Occurrence, Attribute, Value);

**Meta Model(s)**

Dataflow APIs hierarchy:

Roles reify to Meta Meta Model Roles:

(Relation, Statement, Kind, Resource);

(Augmentation, Mapping / Predicate, Transform / Template, Resource);

Dimensional APIs hierarchy:

Roles reify to Meta Meta Model Roles:

(Relation, Statement, Kind, Resource);

(Dimension, Unit, Measure, Value);

Semiotic APIs hierarchy:

Roles reify to Meta Meta Model Roles:

(Relation, Statement, Kind, Resource);

(Context, Sign, Object, Value);

Domain Models. Service APIs. Templates.

Dataflow, Dimensional and Domain Models declaratively stated / reified in root Meta Meta Model Roles.

Messages. Encoding / I/O:

See 'Service Resources'.

(Class, Instance, Member, Value);

Augmentation:

Domain (Templates)

Meta Model (Activation)

Meta Meta Model (Augmentations)

Meta Model (Activation)

Domain (Templates)

Encodings:

(a (b (c (d, nil) : First / Rest binary tree.

(K: (K: inst, V: cls), V: (K: mcls, V: occur))

Object / Occurrence: Reify layers Object / Occurrence keys / values. Reference Model along with OntResource.

Message I/O (see Service Resources):

(Class, Inst, Attr, Val);

Templates: I/O Dataflow (aggregated messages) declarative bindings. Aggregated message facade (Augmentation, render / apply layer entity schema / data). Forms / Flows.

Forms / Flows: HATEOAS HAL. MVC. REST. Meta Models DCI based protocol.

HATEOAS: Forms / Flows Operations / Dataflow Representation / State IO (CRUD) prototypes / templates. Dialog. Prompts. Gestures. Context: navigation state (i.e.: pick operation value prompt shows value type Forms). DDD DOM.

Models Sets Reification (Populate DOMs / FCA):

OntResource

Object

Occurrence

Role : CSPO hiers Sets.

Resource : Role Set member.

Statement : Set members Role aggregation.

Kind: Statement Resources aggregations. Roles intersection sets.

Relation: Kind Statements aggregations. Transform: Kind Resources related to themselves (ID), then Relations to other Resource via Dataflow Kinds domain / range relationship (ordered).

Relation: aggregated aligned entities. Views (transforms). Kind members occurring in Statement Resource(s). Functors / Monads:

Relationship<A : Relation>::flatMap(F : Function<A : Relation, B : Relation>) : Relationship<B : Relation>;

Entity<A : Relationship>::flatMap(F : Function<A : Relationship, B : Relationship>) : Entity<B : Relationship>;

Function: declarative dataflow transform.

Dataflow Kinds domain / range: Grammar. Reify Kinds as CSPO and assert Statement. Aggregate further Kinds (until primitives).

Valid Statement (Grammar / Relation): domain / range, CSPOs backing assertions apply.

I/O: Parse / aggregate input Statements into corresponding Roles / Resources. Aggregate / match Kinds. Relations: render / activate. Resolve output Statements.

Built in Relation(s): ID, equals, inverseOf, parent, child, previous, next, etc. (upper ontology / meta model). Composites: Monad Zippers.

Meta Meta Mode (TBD):

DOM: OntResource, Object, Occurrence, Role. Express Meta Model levels metaclass, superclass, class, instance, occurrence relationships.

(Superclass, Class, Occurrence, Instance);

Metaclass: Role.

Occurrence: Role instance.

**Monadic Functors / Transforms:**

Parsed from / rendered to (materialized) to Meta Meta Model DOM Relations Monad Statements. Inheritance from Relation, Entity, Relationship, Flow.

Domain Models Entities / Relationships / Flows: Model parsed / rendered Relation materialized domain functors / transforms. Monadic layer for functional operation of underlying DOM Model Relations.

Model Relation:

(Relation, Statement, Kind, Resource);

ToDo: Resource selector API. (Relation).

ToDo: Transforms data flow. Application activation (available operations / state signatures / event flows). Browseable DOM / reified transforms (HATEOAS), reactive / event driven dialog / protocol: render applicable transforms.

Monad Transforms (hierarchically implemented):

Type Constructor.

Unit.

Bind.

Map / FlatMap.

CRUD (parametetized flatMap):

Resource::Append(Resource arg) : Function<Resource, Resource>;

Resource::Retrieve(Resource arg) : Function<Resource, Resource>;

Resource::Replace(Resource arg1, Resource arg2) : Function<Resource, Resource>;

Resource::Remove(Resource arg) : Function<Resource, Resource>;

Relations / comparisons:

Resource::Equals(Resource) : Function<Resource, Resource>;

Resource::Contains(Resource arg) : Function<Resource, Resource>;

Others: Primitives, Semiotic, Dimensional, etc. Domains modelled.

Domain Transforms:

Resource : Function (Mapping):

(Resource, Occurrence, Attribute, Value);

ActivationService:

Activation: Layers hierarchy (polymorphism), Layers reification.

Browse Layers contexts for available state transforms. Activation Resource states applicable to next / previous states / productions.

ActivationService Resource API:

Resource::Activate(ctx : Resource) : Resource[];

Resource::Available(Resource) : Resource;

Resource::Apply(Resource) : Function<Resource, Resource>;

DOM Layers Monads:

Resource<OntResource[]>; Aligned Resources (metaclass, class, instance, role, occurrence);

Kind<Resource[]> : Aligned Types.

Statement<Kind[]> : Aligned Statements.

Relation<Statement[]> : Aligned Assertions.

Entity<Relation[]>;

Relationship<Entity[]>;

Flow<Relationship[]>;

Domain<Flow[]>;

Entity Monad. Aligned Resource (data):

Entity<Relation[]>;

Entity<Relation[]> Monad. Relation[]: selector, Relations which are instances of / wrapped by this Entity scope.

Example: Employment anEmployment Entity. anEmployment Relation Statements. Selector matches Employment Relations.

Entity Monad Transforms:

Apply Relation selector / Relation CRUD to Employment anEmployment.

Relationship Monad. Aligned schema:

Relationship<Entity[]>;

Relationship<Entity[]> Monad. Entity[]: selector, Entities which are instances of / wrapped by this Relationship scope.

Example: Employmentship aggregation / concept of Employment Entities. Selector matches Entities Relations.

Relationship Monad Transforms:

Apply Entity selector / CRUD to Employmentship Employments.

Flow Monad. Aligned behavior:

Flow<Relationship[]>;

Flow<Relationship[]> Monad. Relationship[]: selector, Relationships which are instances of / wrapped by this Flow scope.

Example: SalaryRaised Flow for Employmentsip Employments. Selector matches Relationship Entities Relations.

Flow Monad Transforms:

Apply Employmentship selector / CRUD to SalaryRaised Employments.

Domain Monad. Aligned behavior:

Domain<Flow[]>;

**Backend Architecture (To Do):**

XML: XSL, XPath, XLink, XPointer, XQuery, XForms: hypermedia addressing / state / flows encoding / Message endpoints protocol.

Models.

Messages.

Encoding.

Endpoints.

Protocol.

Domain Connectors.

APIs.

Models:

Dispatch to Model layers context resources streams. Message IO. Endpoint Message matches in Model context: activation (Dataflow).

Messages:

Meta Meta Model entities (Relation). Meta Model layers scoped context statements.

Encoding:

Model layers scoped context statements.

Endpoints:

Model layers context resources streams (pub / sub topics). Topics: OntResources(s). Signatures: Dataflow Message IO wiring.

Protocol:

Message Dialog: i.e.: XML encoded Context statements Message IO with Model layer scoped prompts, placeholders, wildcards, variables. Models. Messages. Streams.

Domain Connectors:

Tools. Service Resource: URL, streams (Messages I/O).

Tools:

NakedObjects / Apache Isis. Apache MetaModel. JBoss Teiid. JDBC. JCA. Apache Stanbol. Apache Clerezza. OData / OpenAPI. JSON-LD. Spring HATEOAS / HAL. Apache Any23. D2RQ. R2RML.

Message Matching:

FCA Augmented Models: Context objects / attributes: layer quad resources Role, Kind, Resource.

Resource Context Concept: Resource x Kind.

Resource Context Object: Role x Resource.

Model / Schema Matching:

FCA Resource Context Concept.

Data / Resource Matching:

FCA Resource Context Object.

Matching Productions:

Result Set (query / augmentation result).

Augmentation (aggregation of new statements, alignment of new knowledge, activation of transforms / flows: result set).

Relationship / Entity monads mappings results.

FCA Scaling: Role > Kind > Resource aggregation of matching objects / attributes.

FCA Augment Resources: Role, Occurrence, Object (Resource), Concept, Object, Kind. Grammars. Match schema, instances. Mapping transforms: match behaviors.

FCA Contexts from Sets aggregation:

Set Roles: Context, Subject, Predicate, Object.

Sets aggregation: Statement, Kind (SuperKind, Kind, Attribute, Value) Attributes: class / Values: metaclass, Resource (Meta Model Roles: Kind context, Resource SPO), Context (Relation). Reified Kinds.

Aggregation streams: Sets reactive events aggregation. Sets (ordered) description APIs.

Sets aggregation: FCA Contexts scaled objects / attributes from Sets aggregation. FCAAPI.

**Dataflow. Components:**

Uniform Resource API: Sets, FCA, DOM layers, Monads. Reactive message driven dataflow (topics / signatures).

Inputs. Connectors / Services (active Resource topics).

Sets aggregation.

FCA Scaling. FCA Contexts (layers / occurrences).

DOM Layers / OntResource hierarchy. Augmentation, alignment, activation, matching. FCA alignments (concepts).

(Sets aggregation populates DOM layers FCA augmented or Sets aggregation builds FCA contexts rendered into FCA augmented DOM layers).

Functors. Parse DOM: Instantiate Relationship / Entity Monads (selectors / contexts). Model services interactions renders functors possible transforms as browseable (HATEOAS reified) resources / contexts: reactive dialogs / prompts (HATEOAS / HAL protocols).

Model Services: Browse DOM layers. Monads parsed DOM interactions services (functor contexts) available as operations over rendered models (HATEOAS).

Interactions: Services. Browse DOM. Apply selectors / browse available transforms (Monads / HATEOAS). Monads applications render / update DOM / HATEOAS browsing response.

Outputs. Connectors / Services (active Resource topics). Feedback (Events Inputs).

**Deployment:**

Apache MetaModel. JBoss Teiid. Connectors (I/O). APIs: Model Services (reify data, schema, behavior alignment in Connectors data structures). OpenRefine Knowledge (data, schema, behavior) alignment extensions (Model Services APIs). Knowledge transactions (inferred "wizards") contextual wiki like augmentation: Apache Stanbol (guided assistance). Context Wiki: JCR XPath / Apache Clerezza.

**APIs:**

Contexts (DCI / HAL / HATEOAS):

Context Guided Data augmented (contextual hypermedia) Interactions. Wizards APIs. XForms: rendering (REST HATEOAS).

**Service Resources:**

Service Resources. ContextResource scoped prediction generalizations (encodings):

Meta Meta Model: (Relation, Statement : Entity, Kind : Relationship, Resource : Flow);

(PredictionService : Relation, Context: Entity<PredictionService>, Features : Relationship<Entity>, Output : Flow<Relationship>);

Naming Service:

Synsets, generalization/specialization term rels:

(NamingService, Context, TermRel, Term);

Registry Service:

Hierarchical key / value (property graph):

(RegistryService, Context, Key, Value);

Index Service:

Apache Lucene, Vector Space Model Triple / Quad polygon encoding:

(IndexService, Context, Term, Result);

IOService (Connectors):

(IOServiceConnector : Relation, ContextResource : Entity, Attribute : Relationship, Value : Flow);

Relation: Aligned Context.

Entity: Aligned Resource.

Relationship: Aligned Schema.

Flow: Aligned Behavior (schema resource data flows).

ToDo: Resource selector API.

ToDo: Transforms data flow. Application activation (available operations / state signatures / events). Browse (HATEOAS), reactive / event driven protocol / dialog. Applicable transforms (example: IO Connector updates).

Domain Service (ActivationService):

ActivationService:

Services which performs addresses / body request activations (ResolutionService).

(Domain, Flow : Flow, Role : Relationship, Item : Entity);

Flow: Use Case. Produced / available for Domains.

Role: Prompts. Produces / available in state of Flow. Item: product, prompts consumer / producer kind.

Item: Product / Good / Need (Goal). Produces / populated by Roles (Prompts).

(DomainService : Relation, Resource : Domain, Context : Role / Relationship, Item : Entity);

Retrieves available workflows Items for Roles in domain.

**Upper Ontology. Grammars. Primitives:**

Primitives:

Primitive Resources for primitive Kinds for primitive Statements for primitive Relations.

Primitive relations example: opposite, inverseOf, causeEffect, etc.

Grammars:

Resources productions for Kinds productions for Statements productions for Relations productions.

Upper Ontology: Primitives Grammar.

**Ontology Matching:**

Data, Schema, Behavior matching / alignment.

Data: keys / values.

Schema / Information: relation tuples rows.

Behavior / Knowledge: relation tuples rows data / information flows (dimensional).

FCA Augmented Resources.

See Protocol Addressing / Matching.

**Protocol. Message IO. Dataflow:**

DOM Hierarchy:

Resource<OntResource[]>;

Kind<Resource[]>;

Statement<Kind[]>;

Relation<Statement[]>;

Entity<Relation[]>;

Relationship<Entity[]>;

Flow<Relationship<Entity[]>;

Domain<Flow[]>;

Addressing / Matching:

Message Event dispatch. Message is an address plus browse state representation.

Matching. Distributed Addressing Encoding:

Encoding: Encode metaclass, class, instances, occurrences (contexts) in addresses.

Identifiers assigned according context objects and metadata "paths" following a pattern or "shape" in a way analogies can be inferred (Monad Zippers) of its (nested / linked ) metaclass, class, instances, occurrences "trees" (cons cells).

Matching: Same objects resolve to equivalent addresses in different models when addresses follows / match Zippers shape.

Populate ResolutionService Relations. Zippers paths / shapes specialized / generalized matches (concepts hierarchies). Others services metadata.

ActivationService:

Services which performs addresses / body request activations (ResolutionService).

(DomainService : Relation, Resource : Domain, Context : Role / Relationship, Item : Entity);

ResolutionService (Matching):

(ResolutionService : Relation, Resource : Entity, Model : Relationship, Resource : Flow);

Resulting Flow: next state (activation) over browse representation request.

ActivationService:

Referrer.

Address.

Body.

Next State.

(ActivationContext : Relation / Referrer, Statement : Body, Kind : Address, Resource : Next State);

ActivationService Resource Object (Resource) is next / matching Resource for request context (Referrer, Address, Body).

Activation Patterns (DOM Layers hierarchy matching / Layers reified Resources matching):

(Resource, Role,

(Kind, Resource, Role,

(Statement, Kind, Resource, Role);

(Relation, Statement, Kind, Resource);

(Entity, Relation, Statement, Kind);

(Relationship, Entity, Relation, Statement);

(Flow, Relationship, Entity, Relation);

(Domain, Flow, Relationship, Entity);

Semiotic / Dimensional Roles Facets: Idem for equivalent Relation hierarchy layer.

Monads: Reify available Transforms as activable Resources (Function addresses). REST / HATEOAS HAL.

Prompts / Dialogs: Function arguments (values / options) shown as link addresses in Transforms navigation Flows. Activation browse of Resources in Transform context.

Chained Activations for complete contexts resolution / flows. Complete layers productions rendering / navigation from higher to lower layers.

Encoding / Parsing:

Representation: Message Event production / consumption.

Abstract Content Type: Render Model DOM Hierarchy.

Browse: request address content representation (extracted from current state) embedding current state representation as request context body. Model matches address and returns augmentation using request body as argument / context.

Sets / Individuals Mappings:

IDs: metaclass, class, instance, context, role, occurrence, previous, next ID roles relations for Model Set Contexts.

Augmentations / Transforms: Model / Domains functional mappings. Order. Dimensions. Axes. Flows. Hierarchies. Inference / Population.

Levels: Augmentations. Mappings.

Levels: Resource, Kind, Statement.

Levels: Reify Statement as Kind, Kind as Resource, Resource as Statement.

Levels: Reify Resource as Kind, Kind as Statement, Statement as Resource.

Sets / Individuals Mappings:

Levels (layer statements) shifts (quads matrix). CSPO roles:

(Dimension, Resource, Kind, Statement);

Resource: Semantic Address / ID. Dereference / Browse: Representation State / Roles (CSPO) according referrer / occuurrence / transforms / mappings (monadic pipelines) contexts. Examples: Faceted search (build context template schema roles), Shopping Cart (build use case representation items via state representation exchanges).

**Dataflow. Layers:**

Sets Aggregation:

Sets: Context / Role, Resource, Kind, Statement. From ActivationService / IOService raw RDF / events response statements (feedback). FCA / Transforms synchronization (events / signatures).

Layers: Context / Role, Resource, Kind, Statement, Relation, Entity, Relationship, Flow, Domain. Quad store mappings (functions). Sets / FCA / Transforms synchronization.

FCA Context Aggregation:

Populate concepts objects / attributes from layers mappings. Sets / Transforms synchronization (events / signatures). Sets / DOM / Transforms synchronization.

Resources belonging to multiple sets / concepts: degree of pertenence. Concepts hierarchy specializations (Kinds).

Objects: Layer Context Resources.

Attributes: Layer Contexts scaled to their CSPO (Context, Occurrence, Attribute, Value) Resources.

DOM Functional Aggregation:

Instantiate DOM Layers Monads from FCA Context (Wrapper type: Layer Context Role, values: FCA Objects) / Transforms (FCA Concepts. Matching concepts signatures reified into Resource functions transforms mappings. Concepts lattice flows. Aggregation: lattice concepts (transforms) / objects (functor values) / attributes.

Transforms (Resource functions transforms mappings) populated with possible source / dest values from context concepts objects / attributes. Concepts flows available when Resource matches source attributes. Transform Resources available for each DOM layer.

Example: 'anEnterprise' Entity Monad. Available flow / transform: concept Resource: 'Corporation'. Reified lattice concepts as Resource functions transforms mappings (source / dest mappings).

Transform application merges / translates Entity Relations with applicable mappings from Resource functions transforms mappings (of which concept Resource has source / destination transform mappings).

Example: Entity 'aCorporation', concept Resource 'CEO', Entity with merged attributes from Resource functions transforms mappings.

Protocol. HATEOAS. Available transforms flows rendered as browseable Resources. CRUD: Browsing values for a Resource mapping transform has REST semantics for activating concepts with new Resources.

Selectors: matching / activation: match Monads functors by their attributes (signatures), apply transform (mapping function resource transform request address) over referrer body (yields next state functor). Events: Monad functors listen matching browsing events and publishes transform results (ActivationService IO streams). Update DOM / Layers. FCA / Sets synchronization (events / signatures).

Context Resource functions transforms mappings (FCA Concepts Augmentation Layers):

(Object : Resource, Concept, Attribute : Resource, Value : CSPO Role);

Object Resource Role type: Monad wrapper layer type. Object Resource: Monad values (Resource subjects in concept context).

(Flow, Object : Resource, Concept, Attribute : Resource);

Flow queries / prompts for Attributes. Update context / augmentations.

TBD.

**To Do:**

Context Sets. Layers Aggregation (MapReduce).

Backends (Layers Repository, Services):

Property Graphs. Graph algorithms (encodings). Functional APIs: Event Driven Facade (Reactive Services).

Stores: Jena (Fuseki Services), RDF4J (N3, RIO, SAIL), Neo4j.

Fuseki: SPARQL Services.

Models: RDFS / OWL (Jena). Reasoning.

Schema: SHACL, ShEx. Matching.

Protocols / Formats: XML RDF / OWL, JSON-LD, Turtle, N3.

Endpoints: SPARQL, JSON-LD / HAL, GraphQL. Queries / Templates (HATEOAS HAL Forms / Flows).

FCA: From federated Peers / Endpoints layers navigation representations.

DOM: From FCA Context Lattice. Client high level DOM representation APIs (reactive / streams): navigation, transforms. Client I/O: DOM representations navigation (browse / transforms).

Endpoint DOM representation navigation: Standing in a browsed address body (referrer) render address context: nested context aggregated SPOs (DOM Monad values), reified HATEOAS Transforms as context parent contexts navigations (browsing shows Transform attributes prompts as browseable HATEOAS).

JSON-LD REST HATEOAS / HAL / GraphQL DOM wrapper Endpoints. SOAP Endpoints. APIs / Interfaces (objects / schema / behavior) inferred from DOM models. Discovery (workflow contexts state flows) through DOM metadata.

FCA / VSM (Vector Space Model) Encoding:

Attributes: Resource URIs. Polygon side lengths (class).

CSPO Roles (scaling): polygon sides (metaclass).

CSPO scaling: ordered side position.

Polygon sides dot-notation ordered sides lengths: Resource Layer Statement IDs (instance).

Sides dot-notation sum: side in context (occurrence).

Normalization: Resource URI attributes embeddings / primes quad polygon sides lenghts.

Nested Resource encoded attribute values (layers hierarchy): sides lengths concatenation (ordered dot notation) sum (occurrence).

Graph navigation (layers / transforms: concepts / objects containing / contained in concepts / objects attributes IDs / lengths).

Distributed Contexts (label / tag metadata statements) and Versioning: Blockchain / Git / Apache Kafka persistence. Event sourcing / DIDs. Distributed back ends / data sources.

Dataflow layers:

Sources / Persistence: Reference Model encoding. Event sourcing. Ontology Matching. Resolvable DIDs.

Sets (Layers Roles) Population / Aggregation (Layers: MapReduce).

Triple Store / Model Resource Layers.

FCA Contexts. Encoding. Flows. Order (types: dataflow signatures domain / range, instances: dimensional attributes).

Functional DOM: Meta Model Resource Layers Functional DOM (FCA / Monads / Transforms)..

Forms / Flows: DCI HATEOAS DOM Functional Protocol Client APIs.

Tabular / Object Type Object pattern DOM / JAF APIs. Graph DTOs (REST).

Events: Reference Model encoded streams (Sources to / from APIs / IO).

Meta Model (Layers / DOM):

CellValue

ColumnField

ID : occurrence (PK)

Context : instance (table)

Role : metaclass (CSPO)

Resource : class. Monad Value (instance)

Kind : selector / transform (Functor mapping). Monad Value Type (metaclass / role)

Statement (context)

Relation : Kind Grammar (Productions). Monad Instance (occurrence)

Entity : Kind Grammar (Rules). Monad Type (class)

Relationship

Flow

Domain

Monad and Grammar roles apply for reified layers Resources.

Messages:

Rules: Entity (grammar: infer available messages).

Productions: Relation (infer / parse Messages).

Matching: Kind (selector : Entity Relations).

Transforms: Kind (transform: Relation Productions of Entity Kinds).

Dataflow: Result Transform matching rules signatures.

Layer::flatMap(attr : Kind) : Layer;

Message: Graph layer statements(s) populated with Relation (Productions) nested into Entity Rules to be applied / applicable to the Message Relations (Productions). Relations with concrete Resources or Kind matching model layers instances. Existing or new Production: Resource or empty Kind results. Update / Delete: override previous version.

Build Message graph via navigation of the model (Forms / Flows HATEOAS APIs, Kinds domain / range). Transform mapping: Kind prompts: apply Rule Kind to Production Resource: Productions.

Message: Relation statements. Productions.

Message: Rules. Relation statements of Rule Kinds on to operate over Relations of Kind.

Create: Relation not matching existing one. Rules Kinds Productions populated prompting Message Relations / Model with Rules Kinds.

Update: Update Model Relation Resource matching Message Relation of Rule Kind updating Relation Resource of Rule Kind. Update references. Versioning.

Delete: Relation Kind in Rules but not in Productions. Update references. Versioning.

Retrieve: Relation Resource of Rule Kind matches model Kind Resource. Prompts.

TBD: Monads, Layers, FCA, Sets, Persistence synchronization. Events.

TBD: Encoding, Model, Protocol (Messages DCI Forms / Flows):

Data Model: DOM / JAF. Object / Tabular Type Object pattern. OGM. TMDM.

Reference Model: Cons lists / FCA nested context pairs / contexts (Link Grammar constraints). (Type : Value). TMRM.

To do:

Render synchronization / consistency across dataflow layers via Meta Model DOM / Messages. Reify layers into Meta Model. Resource layer implementations (context URI) invoked / invoking Resources with Resource Message populated (encoding) with event upper layers values.

Upper / Onto Matching: reify Resource upper layers as Resource and aggregate into lower layers. Reified Entity, Relationship, Flow, Domain as upper layers and aggregated downwards (Rules / Productions). Productions dataflow (domain / range).

Ontology Matching:

Matching: Resource occurs as context / occurrence / atribute / value or class / occurrence / context / metaclass / instance in equivalent occurrence contexts (kinds / order / shapes / type hierarchies).

Meta Model encodes mappings for equivalence / relations hierarchies for entities instance occurrences in roles in contexts for concepts recursively till upper onto / primitive terms / relations.

Workflows (Domain Goals) general purpose ontology matching integration framework.

Workflows (Domain Goals) general purpose ontology matching integration framework.

Components:

Reference Model (Component Message Adapters). Component Monads of Component Nodes Functional events (bus) dataflow (selector signatures).

I/O. Persistence. Events. DIDs Components Nodes

Sets Component Node

Layers Augmentation: Aggregation (layers), Alignment (ontology), Activation (dataflow) Component Node

Layers Quads Component Node

Triple Store Component Node

FCA (Monads AST Builder. Updates Quads Productions) Component Node

Layers Monads / Parser Monads (Messages : Rules / Productions). Functional events dataflow (selector signatures : Activation) Component Node

Forms / Flows (Grammar / Protocol Builder. Prompts) Component Node

Augmented Resources Contexts / Interactions Services Component Node

OGM / Client Drivers Services Component Node

Services / Mappings:

Upper Ontology. Grammars. Primitives. Ontology Matching

DOM Hierarchy:

Resource<OntResource[]>;

Kind<Resource[]>;

Statement<Kind[]>;

Relation<Statement[]>;

Entity<Relation[]>;

Relationship<Entity[]>;

Flow<Relationship<Entity[]>;

Domain<Flow[]>;

Meta Model (Layers / DOM):

CellValue

ColumnField

ID : occurrence (PK)

Context : instance (table)

Role : metaclass (CSPO)

Resource : class. Monad Value (instance)

Kind : selector / transform (Functor mapping). Monad Value Type (metaclass / role)

Statement (context)

Relation : Kind Grammar (Productions). Monad Instance (occurrence)

Entity : Kind Grammar (Rules). Monad Type (class)

Relationship

Flow

Domain

**Borrador:**

## **Application:**

Workflows (Domain Goals) general purpose ontology matching integration framework.

The idea is basically that if you have: A) an ERP, B) a CRM, C) Drive or similar and D) an issue tracker / workflow / BPM or similar you see a "syndicated" dashboard that allows you to browse "annotated" hypermedia aligned and matched about the state of the processes in each of them.

After that if you act in A certain behavior the relevant / entailed changes are reflected in B, C and D (according to the processes and flows of the business domains of the applications that you have integrated) and that, through a “virtualized” model, have a homogeneous / aligned view (google JBoss Teiid, as an eventual back end, for example) of the domains you have.

All this through a back end of graphs (triple store), functional programming, "ontology matching", inferences and even Machine Learning in layers from the model to the functional view and various types of clients / APIs of applications or services.

Then there was the issue to export to a "standard" format (WS- \* / StratML, Swagger, GraphQL or an "Object Graph Mapper") the metadata that allows to consume or create endpoints / services / clients "declaratively" on the "inference" of what would be the APIs or interfaces and their flows (google REST HATEOAS or REST HAL) from what was integrated. All this goal / purpose driven: metadata about the processes / domains and their objectives.

There are particular cases (WS- \* / StratML, for example) that would serve both as "input" for the description of domains orchestration and for formatting the "output" of some type of description / interfaces.

### **Features:**

Data, Information, Knowledge exchange: data / schema / behavior Augmentation of virtualized and syndicated / aligned business domains. Business domains applications purpose / problem "spaces" interactions / translations. Addressable interactions: event sourcing. Purpose modelling: Business Domains.

Products And Services Community Exchange Network. Resource Oriented Knowledge Computing. Purpose driven Needs / Goods / Products Goals interactions.

Semantic Hypermedia Browser: declarative front-end / services. Forms / Flows. Annotation, Augmentation & other Domains.

Domains Workflow. Layer Contexts Domains. Domains Workflows (Domain Goals) general purpose ontology matching integration framework.

Levels: Upper / Onto Matching: reify Resource upper layers as Resource and aggregate into lower layers. Reified Entity, Relationship, Flow, Domain as upper layers and aggregated downwards (Rules / Productions). Productions dataflow (domain / range). From UI Gesture to backend operations.

### **Domain Component Model:**

Runtime configured (model: triple store, controller: object layer, view: functional layer) for Message parsing and Augmentations executions.

#### **Model:**

Triple Store. Meta Model Schema (RDF / RDFS). Upper Ontology. Primitives.

**Meta Model:**

Layers class hierarchy. Relationship : Relation / Relation kindOf Relationship.

Layer instances, classes, metaclasses, contexts, occurrences shape mappings (layers transforms / message augmentation templates) grammar shapes.

Reified Statement (Relation) Mappings: Model shapes / grammar metadata. Augmentations reify to / from meta model mappings / ontology statements.

Instance : URI

Class : Instance

Context : Class

(Context, Class, Instance);

Metaclass (Kind / Role) : Context

(Metaclass, Context, Class, Instance);

Occurrence : Metaclass

(Occurrence, Metaclass, Context, Class);

Statement (Relation) : Occurrence

(Statement, Occurrence, Metaclass, Context);

Relationship : Relation

Flow : Relationship

Domain : Flow

Model : Domain

##### **Meta Model Quads schema:**

URI (Predicate).

Statement : URI (Mapping).

(URI, URI, URI, URI);

Value : Statement (Function).

(Value, Occurrence : URI, Attribute : URI, Value : URI);

Sign : Value

(Sign, Value, Occurrence : URI, Attribute : URI);

Object : Sign

(Object, Sign, Value, Occurrence : URI);

Context : Object

(Context, Object, Sign, Value);

Kind : Context

(Kind, Context, Object, Sign);

Resource : Kind

(Resource, Kind, Context, Object);

Relation : Resource

(Relation, Resource, Kind, Context);

Relationship : Relation

(Relationship, Relation, Resource, Kind)

Flow : Relationship

(Flow, Relationship, Relation, Resource);

Domain : Flow

(Domain, Flow, Relationship, Relation);

Model : Domain

(Model, Domain, Flow, Relationship);

Data Layers:

Context Layer (reified) aggregation / population to / from aggregated / populated Object, Sign, Value layers (Data layers) Ontology (upper) matching: Predicates / Mappings / Functions. Contexts matching Occurrences / Attributes / Values.

Schema / Information Layers:

Kind, Resource, Relation Aggregated / Aligned Layer contexts instances.

Behavior / Knowledge Layers:

Relationship, Flow, Domain Aggregated / Aligned Layer contexts instances.

Domains Context Services:

REST / HATEOAS / HAL Context / URI backend API interface. Dataflow Context streams (domain / range). Layer Context type / instance type (service selector) events (Message) driven interface Augmentation (service features) domain streams signature request / response. Integrated Models (domain / services) participate in dataflows. Message Resolution: Model (integrated / contexts), Message prompts, dataflow domain context service / URI. Sync Model Contexts (i.e.: response matching domain / range of persistence contexts / services.

* Hierarchy: upper layers reify / render lower layers. Augmentations populate / aggregate lower / upper layers. Example: Domain Service populates data layers from context layers aggregations / aggregates Context instances from data layers.
* Message matching flows from more specific Layer instances (Statement predicates: Model onwards, URIs if none matchs / general upper ontology reified layers defaults) until final Context instance found with matching Message Predicates (selector: layer / type / instance). Matching Context Layer instance SPOs: aggregate / augment more general Layers Contexts instances. Matching Context Layer instance CSPs: aggregate / augment more specific Layers Contexts instances. Layers Context types aggregation signatures.
* Domain services dataflow: Contexts instances invoked according domain / range signatures. Events: LayerType::onMessage (matches / next layer). Traversal aggregates matchings from previous layers until aggregated Contexts.
* Context: Layer type / instance matching selector Context. Domain Services invoked passing Message as argument:
* Object : yields Aggregation statements. Contexts / Occurrences domain / range.
* Sign : yields Activation statements. Occurrences / Kinds domain / range.
* Value : yields Alignment statements. Kinds / Resources domain / range.
* Domain Service behavior: Context instance URI APIs. IO: Message. Domain features / busines logic (express declaratively: TBD). Process invocation result Message(s) recursively (dataflow). Hierarchy Context matching: reified URIs / Contexts.

#### **Controller:**

Resource Layers object hierarchy API.

Controller: Resource Layers object hierarchy API. Named Transforms (Resource URI Service interface / implementation bindings) dataflow: signatures pipelines. Triple Store object graph (DTOs).

Functional Layers Domain model / transforms (events / controller). Named Transforms (Resource URI Service interface / implementation bindings) dataflow: signatures pipelines.

##### **Triple Store object graph (DTOs / OGM):**

URI

Statement

Value

Sign

Object

Context

Kind

Resource

Relation

Relationship

Flow

Domain

Model

Controller: DTOs:

Aggregated super Contexts occurrences collection.

Statements expanded view (occurrences traversal).

Transforms:

DTO: Mapping (Function). From DTO Layer type to DTO Context Layer occurrences type. Stateful Transforms.

TBD.

#### **View:**

Layers Domain hierarchy (Functional API).

Monads AST / Parser Builder. Monads: Parsing / Matching, Zippers. Introduction. API: Augmentations, Transforms / Mappings. Traversal. Dataflow.

Layers Monads / Parser Monads (Messages : Rules / Productions). Functional events dataflow (selector signatures : Layer instance Activation).

Domain Declarations: populate layers from Semiotic Context layer Domain description layers resources: Service Resources I/O layers matching / producing semiotic statements for Domain I/O.

##### **DOM / AST Hierarchy:**

URI<URIDTO>;

Statement<StatementDTO>;

Value<ValueDTO>;

Sign<SignDTO>;

Object<ObjectDTO>;

Context<ContextDTO>;

Kind<KindDTO>;

Resource<ResourceDTO>;

Relation<RelationDTO>;

Relationship<RelationshipDTO>;

Flow<FlowDTO>;

Domain<DomainDTO>;

Model<ModelDTO>;

Functional Model Monads wraps Meta Model Layers DTOs which represents an endpoint / interface for its Resource URI type and instance.

View. Monads:

Layer Monad::of(Layer Monad hierarchcally compatible DTO);

Layer Monad::flatMap(Layer hierarchically compatible DTO) : DTO Context Layer occurrences type Layer Monad.

Layer Monad::value : DTO;

### **Augmentations:**

Domain Controllers (DTO) handled Functional Model API Selector / Transforms.

Matching: Selectors.

Traversal: Layers Contexts.

Matching / Traversal: Resource DTO implements Role / Predicate (Role / Predicate Function / Mapping). Aggregated Resources set specification : Message (model / input prompts).

Transforms: Resource DTO implements Mapping / Function.

Augmentations:

Aggregation: Occurrences (clustering). Flat Map of current DTO type.

Activation: Attributes (classification). Flat Map of lower layer DTO type.

Alignment: Values (regression). Flat Map of upper layer DTO type.

TBD.

* Model: higher Model / Functional layer.
* Augmentation Domain:
* Message:
* (Selector : Statement, Aggregation : Statement, Activation : Statement, Alignment : Statement);
* Model (DTOs). Addressing (visitor):
* Model::onMessage(msg : Message) : Monads (Alignment stream)
* Augmentations (Monads);
* Monad::flatMap(dto : DTO) : Monad : functional / model hierarchies / aggregation;
* Browse / Traversal : Messages.
* Augmentation Domain Statements parses / emits Model / Message Statements.
* Aggregation Domain:
* (Domain, Layer, Rule, Occurrences layers);
* Activation Domain:
* (Domain, Occurrence, Rule, Attributes layers);
* Alignment Domain:
* (Domain, Attribute, Rule / Occurrence, Value / populated Layer statement);
* Dataflow.
* DTOs: Model / OGM. Mappings. Functions. Predicates.
* Monads: Functional View.
* Message: Controller / Command Transform API (Transforms Selector / Augmentations Roles / Predicates). Roles / Predicates encoding / addressing resolution.
* Selector: Monads from Selector Transform Role (stream). Role predicates matching / addressing Resources (Model visitor). Aggregated Message Resource set Monads (i.e.: Relationships / Relations). Layers Context instances selector.
* Selector: Select Context Layer instance.
* Aggregation: Rules / Productions. Monads from Aggregation Transform Role (stream): Flat map on Message Aggregation Role / Predicate over Selector stream. (i.e.: Flow Contexts having Relationship Occurrences having Relation Occurrences).
* Aggregation: Context instance types Aggregation criteria (Role / Predicates; filter / join / prompt: Model / Message deferred resolution). Aggregation: (Context, Occurrence, Attribute, Value) mappings / axes aggregation predicates.
* Aggregation: Example: Criteria for which a Statement instance is occurrence in a (new aggregated) Relation instance and a Relation instance is occurrence in a (new aggregated) Relationship instance. OGM (DTOs) and Context instance types (RDFS). Contexts types schema and instance types schema / shapes.
* Aggregation: Assert Context Layer instance occurrences in next Context Layer (recursively). Inheritance levels polymorphic Aggregation (i.e.: aggregate Relationship as a Relation).
* Aggregation: Shapes / Parsing / Traversal: Monad Zippers / Predicates.
* Activation: Layers Contexts types instance types. Context occurrences (aggregation) Kinds.
* Activation: Types / Attributes Monads from Activation Transform Role (stream): Flat map on Message Activation Role / Predicate over Aggregation stream. Aggregated statements instance type Attributes. Statement predicates: previous context statements subjects (filter / join / prompt: Model / Message deferred resolution).
* Alignment: Attributes / Values. Monads from Alignment Transform Role (stream): Flat map on Message Alignment Role / Predicate over Activation stream. Activated statements Attributes Values. Statement objects: previous context statements predicates (filter / join / prompt: Model / Message deferred resolution).
* Dataflow: Domains Layers Mappings. Further Augmentations (Messages). Activation values matches Domain subject signatures. Domain activation Layer emits new Resource set Messages. Augmentations according Model / Domain.
* Augmentation Domain Statements. Model Messages: Core Augmentations. Invoked after each Model Message processing. Layers. Initially matches Upper Ontology / Core Domains.

#### **Aggregation:**

Clustering stream. Registry. Attribute Roles in Contexts. Populate schema quad layers.

Productions: Layers down through the contexts hierarchy are "productions" of previous layers.

Rules: Layer contexts aggregates previous layers contexts as their subjects matching / aggregating same subjects / predicates / objects.

Layers conform a hierachy of which Value is root and Domain is the last Layer in the inheritance chain.

Layers Resource Context / DOM API. Levels (inheritance hierarchy reification). Upper / Lower Layer Roles. Transforms. Bindings (contexts resolution by reference model matchings).

#### **Activation:**

Classification stream. Naming. Attribute Types in Contexts. Populate model Kinds.

Kinds: (Context : Kind, Resource, Attribute, Value);

Hierarchies: Resource reified Kind as Kind Resource (sub Kind).

Kinds layouts:

(S, P): OK; (P, O): SK, (O, S): PK; (SK, OK): CK;

Role: Sets. Layers CSPO Resource types.

Type Promotion (roles). Order (dataflow). Reified Relation / Relationship (Production / Rule) context roles / interactions. Matching.

#### **Alignment:**

Regression stream. Index. Attribute Values in Contexts. Align / complete missing information.

Model reification: Role Context. Addressable Augmentations (Object extension which is result of Context intension).

Kinds: Streams of corresponding Roles.

Semiotic Layer: (Augmentation, Subject, Predicate, Object);

Reference Model: Map Reduce. Reified Layers. Levels.

### **Message I/O:**

Reactive event driven Message Roles / Predicates matching / processing / emission of aggregated matching Resource set results.

Message: Role / Predicate Context matching. Resource set specification.

Parse Message as corresponding Layer Context DTO / Functional wrapper and filter interaction Resource set according Role / Predicate matching. Message resolves to inputs (model prompts) and parameters (client prompts) to be populated and augmented from initial matching layer instance and is populated back with an aggregated response of which augmentations gave as result.

Message Role / Predicate matching Resource set: Perform Augmentations as with source Domains data (inputs / parameters) of new (prompts) / existing Contexts / Occurrences (Aggregation), of new (prompts) / existing Attributes (Activation) and of new (prompts) / existing Values (Alignment) until there are no further Augmentations (dialogs).

Selectors: DTOs implementing Predicate interface.

Transforms: DTOs implementing Function interface.

Streams:

Kinds. Filter: Predicate.

Contexts, Occurrences, Attributes, Values: Productions of Predicates according CSPO role application.

Dataflow: Result Transform matching rules signatures of Semiotic graph domain / range.

Build Message graph via navigation of the model (Forms / Flows HATEOAS APIs, domain / range dataflows). Transform mapping: Message prompts: resolve from model / prompt client.

TBD.

* Model: higher Model / Functional layer.
* Augmentation Domain:
* Message:
* (Selector : Statement, Aggregation : Statement, Activation : Statement, Alignment : Statement);
* Model (DTOs). Addressing (visitor):
* Model::onMessage(msg : Message) : Monads (Alignment stream)
* Augmentations (Monads);
* Monad::flatMap(dto : DTO) : Monad : functional / model hierarchies / aggregation;
* Browse / Traversal : Messages.
* Augmentation Domain Statements parses / emits Model / Message Statements.
* Aggregation Domain:
* (Domain, Layer, Rule, Occurrences layers);
* Activation Domain:
* (Domain, Occurrence, Rule, Attributes layers);
* Alignment Domain:
* (Domain, Attribute, Rule / Occurrence, Value / populated Layer statement);
* Dataflow.
* DTOs: Model / OGM. Mappings. Functions. Predicates.
* Monads: Functional View.
* Message: Controller / Command Transform API (Transforms Selector / Augmentations Roles / Predicates). Roles / Predicates encoding / addressing resolution.
* Selector: Monads from Selector Transform Role (stream). Role predicates matching / addressing Resources (Model visitor). Aggregated Message Resource set Monads (i.e.: Relationships / Relations). Layers Context instances selector.
* Selector: Select Context Layer instance.
* Aggregation: Rules / Productions. Monads from Aggregation Transform Role (stream): Flat map on Message Aggregation Role / Predicate over Selector stream. (i.e.: Flow Contexts having Relationship Occurrences having Relation Occurrences).
* Aggregation: Context instance types Aggregation criteria (Role / Predicates; filter / join / prompt: Model / Message deferred resolution). Aggregation: (Context, Occurrence, Attribute, Value) mappings / axes aggregation predicates.
* Aggregation: Example: Criteria for which a Statement instance is occurrence in a (new aggregated) Relation instance and a Relation instance is occurrence in a (new aggregated) Relationship instance. OGM (DTOs) and Context instance types (RDFS). Contexts types schema and instance types schema / shapes.
* Aggregation: Assert Context Layer instance occurrences in next Context Layer (recursively). Inheritance levels polymorphic Aggregation (i.e.: aggregate Relationship as a Relation).
* Aggregation: Shapes / Parsing / Traversal: Monad Zippers / Predicates.
* Activation: Layers Contexts types instance types. Context occurrences (aggregation) Kinds.
* Activation: Types / Attributes Monads from Activation Transform Role (stream): Flat map on Message Activation Role / Predicate over Aggregation stream. Aggregated statements instance type Attributes. Statement predicates: previous context statements subjects (filter / join / prompt: Model / Message deferred resolution).
* Alignment: Attributes / Values. Monads from Alignment Transform Role (stream): Flat map on Message Alignment Role / Predicate over Activation stream. Activated statements Attributes Values. Statement objects: previous context statements predicates (filter / join / prompt: Model / Message deferred resolution).
* Dataflow: Domains Layers Mappings. Further Augmentations (Messages). Activation values matches Domain subject signatures. Domain activation Layer emits new Resource set Messages. Augmentations according Model / Domain.
* Augmentation Domain Statements. Model Messages: Core Augmentations. Invoked after each Model Message processing. Layers. Initially matches Upper Ontology / Core Domains.

### **Component Domains:**

Functional event driven Domains configures models behaviors. Domain types: service resources, interfaces (transforms), signatures (dataflow).

I/O / Persistence Domain.

Sets Augmentations Domain.

FCA Augmentations Domain.

Endpoints I/O Domain.

Predictions Domain.

Dimensional Domain.

Registry Domain.

Index Domain.

Naming Domain.

Business Domains: business specific domain types.

#### **I/O / Persistence Domain:**

Events (event sourcing). Backends. Peers. DIDs.

Semiotic (Functional Message Signature): (PersistenceType, PersistenceSubject, PersistenceMember, PersistenceValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain persistence resource types (employee).

Member Kind: persistence members resource types (employee/salary;ARS).

Value Kind: range resource types (salary;ARS).

Reify Persistence semiotic predicates as Relationship Relations (Values as Relation Resources). Align domain / range with domains / primitive types (Member Kind, salary;ARS).

Event sourcing:

(PersistenceContext, PersistenceContext, PersistenceMember::new, PersistenceSubject);

(PersistenceContext, PersistenceContext, PersistenceMember::delete, PersistenceSubject);

(PersistenceContext, PersistenceSubject, PersistenceMember::delete, PersistenceSubject);

#### **Sets Augmentation Domain:**

Semiotic mappings population. Augmentations: Aggregation (layers), Alignment (ontology), Activation (layers dataflows). Render Resource hierarchies.

Semiotic (Functional Message Signature): (SetContext, SetParent, SetSubject, SetChildren);

Context Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Parent Kind: domain resource types (resource).

Subject Kind: attribute resource types (resource/resource).

Children Kind: range resource types (resource).

Reify Sets semiotic predicates as Relationship Relations (Values as Relation Resources).

Sets API: Augmentations, Transforms / Mappings. Traversal (Context, Statement, Kind, Resource).

#### **FCA Augmentations Domain:**

Semiotic mappings population. Augmentations: Aggregation (layers), Alignment (ontology), Activation (layers dataflows). Objects / attributes objects / concepts traversal. Render Resource hierarchies.

Semiotic (Functional Message Signature): (FCAContext, FCASubject, FCAAttribute, FCAValue);

Context Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain resource types (resource).

Attribute Kind: attribute resource types (resource).

Value Kind: range resource types (resource).

Reify FCA Context semiotic predicates as Relationship Relations (Values as Relation Resources).

FCA API: Augmentations, Transforms / Mappings. Traversal (Concepts, Objects, Attributes, Products).

FCA / VSM (Vector Space Model) Meta Model Context Encoding:

Attributes: Resource URIs. Polygon side lengths (class).

CSPO Roles (scaling): polygon sides (metaclass).

CSPO scaling: ordered side position.

Polygon sides dot-notation ordered sides lengths: Resource Layer Statement IDs (instance).

Sides dot-notation sum: side in context (occurrence).

Normalization: Resource URI attributes embeddings / primes quad polygon sides lenghts.

Nested Resource encoded attribute values (layers hierarchy): sides lengths concatenation (ordered dot notation) sum (occurrence).

Graph navigation (layers / transforms: concepts / objects containing / contained in concepts / objects attributes IDs / lengths).

FCA Contexts. Encoding. Flows. Order (types: dataflow signatures domain / range, instances: dimensional attributes).

#### **Endpoints Domain:**

Streaming I/O Dataflow.

Semiotic (Functional Message Signature): (EndpointContext, EndpointSubject, EndpointRequest, EndpointResponse);

Context Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain resource types / referrer (employment).

Request Kind: request resource types (person).

Response Kind: response range resource types (employee).

Reify Endpoint semiotic predicates as Relationship Relations (Values as Relation Resources).

Augmented Resources Contexts / Interactions Services.

Forms / Flows (Grammar / Protocol Builder. Prompts). Resource augmentation endpoints. Forms / Flows browsing APIs. DCI: Declarative Forms / Flows.

OGM / Client Drivers Services.

REST: Current / referrer. Rel. HREF. Link body. Metadata. Endpoint Domain.

Monads: Reify available Transforms as activable Resources (Function addresses). REST / HATEOAS HAL.

Prompts / Dialogs: Function arguments (values / options) shown as link addresses in Transforms navigation Flows. Activation browse of Resources in Transform context.

Chained Activations for complete contexts resolution / flows. Complete layers productions rendering / navigation from higher to lower layers.

Browse: request address content representation (extracted from current state) embedding current state representation as request context body. Model matches address and returns augmentation using request body as argument / context.

#### **Predictions Domain:**

Semiotic (Functional Message Signature): (PredictionType, PredictionSubject, PredictionItem, PredictionValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain resource types (image).

Item Kind: prediction resource types (image/face).

Value Kind: range resource types (face).

Reify Prediction semiotic predicates as Relationship Relations (Values as Relation Resources).

#### **Dimensional Domain:**

Semiotic Layer: (DimensionType, DimensionSubject, DimensionItem, DimensionValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O. Time example (contains / before).

Subject Kind: domain resource types (hour; dayOfWeek) : 1. Monday.

Item Kind: dimensional resource types (hour/minutes; dayOfWeek/dayOfWeek) relations: contains / before.

Value Kind: range resource types (minutes; dayOfWeek) : 60. Tuesday.

Data / Key Value: Price.

Information / Schema: Tuples. Price variation.

Knowledge / Behavior: Values relations. Monthly price increase.

Upper Ontology: relations / primitives.

Reify Dimension predicates as Relationships Relations (Values as Relation Resources).

Example: Marriage.

Predicates:

:aHusband :marriedWith :aWife

:marriedWith rdfs:domain :Male

:marriedWith rdfs:range :Female

Relationship:

(aMarriage : Relation, anStatement : marriageStatement, aKind : husbandRole, aResource : aHusband);

(aMarriage : Relation, anStatement : marriageStatement, aKind : wifeRole, aResource : aWife);

(Marriage : Relationship, Marriages : Relation, anStatement : marriagesStatements, aKind : marriageRole);

Predicates / Relationships, Relationships / Predicates entailment. Dimensional: inference / relation types / restrictions.

Encode order / hierarchies / relations (parent / child, prev / next, etc.) / iterations / conditionals / jumps.

Dimensional Domain: dimensions, units, measures, values. Comparisons, relations. State. Events (marriage example). Verbs (action, passion, state). Order (data / schema / behavior).

#### **Registry Domain:**

Key / Value for graph contexts, nodes, predicates.

Semiotic (Functional Message Signature): (RegistryType, RegistrySubject, RegistryKey, RegistryValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain node resource types (person).

Key Kind: registry resource types (person/age;int).

Value Kind: range value resource types (age).

Reify Registry semiotic predicates as Relationships Relations (Values as Relation Resources). Align domain / range with primitive types (Key Kind, age;int).

#### **Index Domain:**

Indexing of graph contexts, nodes, predicates.

Semiotic (Functional Message Signature): (IndexType, IndexTerm, IndexScope, IndexValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Term Kind: domain node resource types (resource).

Scope Kind: dimensional resource types (resource/resource).

Value Kind: range value resource types (resource).

Reify Index semiotic predicates as Relationships Relations (Values as Relation Resources).

#### **Naming Domain:**

Terms translation in contexts for graph contexts, nodes, predicates. Alignment / matching.

Semiotic (Functional Message Signature): (NamingType, NamingSubject, NamingContext, NamingValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain node resource types (term).

Context Kind: dimensional resource types (term/term).

Value Kind: range value resource types (term).

Reify Naming semiotic predicates as Relationships Relations (Values as Relation Resources).

### **Domains Dataflow:**

Layers Dataflow: Augmentation. Rules / Productions matching (Reference Model / Kinds Aggregation).

Semiotic Dataflow: Object Kind matches Subject Kind of Context Kind signatures. (Sucessive Layers Dataflow).

### **Ontology Matching:**

Upper Ontology. Grammars. Primitives.

Matching: Resource occurs as context / occurrence / atribute / value or class / occurrence / context / metaclass / instance in equivalent occurrence contexts (kinds / order / shapes / type hierarchies).

Meta Model encodes mappings for equivalence / relations hierarchies for entities instance occurrences in roles in contexts for concepts recursively till upper onto / primitive terms / relations.

Reify relation from / to predicates (semiotic) / relation entity (expanded relation entity / roles statements). Dimensional measures / state events. Shapes: transforms / rules.

FCA Ontology Matching: Upper ontology / primitives. Reference Model objects / attributes encoding. Encoding (scaling): lattice concepts relations / transforms traversal.

Semiotic mappings population. Augmentations: Aggregation (layers), Alignment (ontology), Activation (layers dataflow transforms: context products).

Reference Model Contexts.

Meta Model Layers Contexts.

Alignments (Reference Model types / values):

Data Alignment: key / val.

Schema / Information Alignment: tuples.

Behavior / Knowledge Alignment: dimensional.

### **Appendix:**

#### **Reference Model:**

##### **Encodings:**

(Type, Object) Key / Value.

(Context, Subject);

(Subject, Predicate);

(Predicate, Object);

Reference Model: Key / Value de-referenceable (for matching / embedding purposes) URIs having as host the peer that identified the Resource. DIDs resolution. Cons cells encoding for Domain Component Model I/O.

Quads Encoding: (Context, Occurrence, Attribute, Value). Kinds. OGM. Sets / FCA. Context, Metaclass / Role, Class, Instance, Occurrence (Context).

Matching: Addressable / Browseable Encodings (FCA / Sets) / Identifiers. Order, Dimensional / hierarchical relations (attribute sets). Root Layers (Reference Model) traversal.

Functional: Selectors (TBD)

#### **FCA Domain Contexts:**

Semiotic (Functional Message Signature): (FCAContext, FCASubject, FCAAttribute, FCAValue);

##### **Reference Model Context:**

FCA Lattice (concepts / objects / attributes): (types / values) x (types / values). Encoding (IDs): ontology matching enabling type / instance calculations / traversal / transforms.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Value | Value | Value | Value |
| Object | X |  |  |  |
| Object |  | X |  |  |
| Object |  |  | X |  |
| Object |  |  |  | X |

##### **Meta Model Context:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Resource A | | | |
|  | Context | Subject | Predicate | Object |
| Resource B | X |  |  |  |
| Resource C |  | X |  |  |
| Resource D |  |  | X |  |
| Resource E |  |  |  | X |

Encoding: FCA Scaling. FCA Context objects and attributes are corresponding CSPO Contexts types scaling enclosed Context types instances. A potential encoding of axes objects and attributes (rows and columns) would be a bitstring of length 4 x n, being n the length of an instance identifier for each quad Context encoded in its corresponding bitstring quad space (4 is for CSPO quad types instances identifiers segments). Then, navigation should be allowed from a pair of object / attribute to another object / attribute: (type, object) x (type, object): (type, object).

##### **Layers Context:**

For each layer context statement build tables which axes correspond to each context CSPO context types. Aggregate CSPO types / values in the form shown below (nested contexts).

Values intersections are instances of corresponding types. Example:

(RoleA x StatementB: KindC) : Kind RoleA plays in StatementB.

Rules are of the form:

(TypeA, TypeB) > AggregatedKindResources;

For example, in Relation lattice:

(someRoleA, someKindB) > AggregatedRelationResources;

Relation matrix:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Relation | Kind | Role | Statement |
| Relation | Relation | Kind | Role | Statement |
| Kind | Kind | Relation | Statement | Role |
| Role | Role | Statement | Relation | Kind |
| Statement | Statement | Role | Kind | Relation |

Matrices for other layers (Kind, Role, Statement, Resource, Context) contexts follows the same principles.

Nested Contexts:

Reifying one aggregated layer SPO layer (for example: Kind in the previous table) has original context matrix axes in the corresponding SPO layer (Subject in this case):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Kind | Relation | Statement | Role |
| Kind | Kind | Relation | Statement | Role |
| Relation | Relation | Kind | Role | Statement |
| Statement | Statement | Role | Kind | Relation |
| Role | Role | Statement | Relation | Kind |

The purpose of this is to retrieve enough concepts (FCA) metadata to populate concepts / objects / attributes conforming a Lattice of related Resources and those relations values (as in the above example).

Layout: The aggregated statements have as Contexts the occurring SPOs in a Context layer statement and its SPOs are the occurrence Context and the other SPOs in the occurring statement. For a Context in an aggregated statement occurring as (SPO) in the occurrence statement, occurrence statement Context is its (SPO) and its aggregated (SPO) is occurrence statement (SPO).

Layout: Having a Context layer, a matrix (FCA context) of the form (CSPO x CSPO) is built for aggregation of models. The aggregated statements (rows / columns) have as Contexts the CSPOs (occurrences) of an axis and and its SPOs are given from the types / values of the context layout. For an aggregated Context statements / matrix, the original context is located in the (SPO) axis from the (SPO) which it was taken from the original Context matrix.

Augmentations: TBD.

Aggregation:

Activation:

Alignment:

Transforms (products / encoding) Dataflow:

Use cases:

Use FCA Lattice for sorting / ontology matching / augmentations / query / ontology browsing.

Aggregation: Complete contexts objects / concepts / attributes by FCA / inference.

Inference example: (Statement x Statement): Relations between both Statements.

Learning: ML embeddings for types / values / concepts.

TBD: (metaclass, class, occurrence, instance) relations / atttributes.

TBD: Set oriented intension (C) / extension (O) and relations between sets.

TBD: Discover IDs / encoding techniques enabling algorithmic translation of models operations.

#### **Sets Domains:**

Semiotic (Functional Message Signature): (SetContext, SetParent, SetSubject, SetChildren);

Augmentations:

Aggregation:

Kinds: (Context : Kind, Resource, Attribute, Value);

Hierarchies: Resource reified Kind as Kind Resource (sub Kind).

Kinds layouts:

(S, P): OK; (P, O): SK, (O, S): PK; (SK, OK): CK;

Role: Sets. Layers CSPO Resource types.

Layers Aggregation:

Productions: Layers down through the contexts hierarchy are "productions" of previous layers.

Rules: Layer contexts aggregates previous layers contexts as their subjects matching / aggregating same subjects / predicates / objects.

Activation:

Resource Context: Kind.

Alignment: Kind Attribute / Value Resource Statements. Shapes (inference of Attribute Value by context, class, metaclass, instance occurrences.

Transforms. Dataflow.

Sets Layout and encoding bitstring mask format:



#### **To Do / Items:**

* Test Browser (Forms / Flows HATEOAS debug console):
* Facets: Browse / Stream Model.
* Model: Browse / Stream.
* Messages: Build / Augmentation Response (Browse / Stream Model).
* Patterns:
* Master / Detail.
* Inheritance.
* Categories. Ordered Sets. Order relation. Inclusion properties / relations.
* Item: Inventory / Product. Order / LineItem.
* Others.
* Patterns: Alignment.
* Pizza ontology:
* For each ingredient calculate total price.
* For average sales calculate ingrediens orders amount, actual / projected revenues.
* Architecture:
* Context instance of Layer type and of Layer instance type (Relation / Marriage). Metaclass, class, instance, contexts / occurrences, roles, attributes, values members from Model.
* Domain I/O: URI REST / HATEOAS / HAL URI API interface (OGM / Domains / IO). TBD.
* DTOs OGM Interfaces:
* URI : Predicate
* Statement : URI. Mapping (Matching URIs). Occurrences.
* Value : Statement. Function (Contexts Subjects matching Predicates). Contexts.
* Resource: Value. Aligned URIs.
* Context: Aligned Statements. Domain: map selector, map aggregation, map activation, map alignment.
* StatementMonad::of(domainUri : URI);
* StatementMonad::flatMap(stmt : Statement) : StatementMonad<Statement> (model / message / backend I/O prompts).
* Message:
* Message (URIs quad) flows from upper layer (URI Statements which reifies all aggregated / sub context types) matching downwards until finding correct context instance.
* Statement Context domains perform relevant contexts streams / I/O (model / message / backends prompts).
* Message:
* (Context / Selector, Occurrence / Aggregation, Attribute / Activation, Value / Alignment);
* Domains (Contexts Mappings / Transforms):
* Domain / mapping function / inference / IO invocation. Dataflow (domain / range).
* LayerMonad.of(LayerContextDTO);
* Flat Map: DTO Mapping Transform.
* Model::flatMap(Selector)
* flatMap(Aggregation) : Aggregated Occurrences.
* flatMap(Activation): Occurrence Attributes.
* flatMap(Alignment) : Attribute Values.
* Data layer: Context Layer (reified) aggregation / population aggregates / populates Object, Sign, Value layers (Data layers) Ontology (upper) matching: Predicates / Mappings / Functions. Contexts matching Occurrences / Attributes / Values.
* Schema / Information Layers:
* Kind, Resource, Relation Aggregated / Aligned Layer contexts instances.
* Behavior / Knowledge Layers!
* Relationship, Flow, Domain Aggregated / Aligned Layer contexts instances.
* Domains Context Services:
* REST / HATEOAS / HAL Context / URI backend API interface. Dataflow Context streams (domain / range). Layer Context type / instance type (service selector) events (Message) driven interface Augmentation (service features) domain streams signature request / response. Integrated Models (domain / services) participate in dataflows. Message Resolution: Model (integrated / contexts), Message prompts, dataflow domain context service / URI. Sync Model Contexts (i.e.: response matching domain / range of persistence contexts / services.
* Hierarchy: upper layers reify / render lower layers. Augmentations populate / aggregate lower / upper layers. Example: Domain Service populates data layers from context layers aggregations / aggregates Context instances from data layers.
* Message matching flows from more specific Layer instances (Statement predicates: Model onwards, URIs if none matchs / general upper ontology reified layers defaults) until final Context instance found with matching Message Predicates (selector: layer / type / instance). Matching Context Layer instance SPOs: aggregate / augment more general Layers Contexts instances. Matching Context Layer instance CSPs: aggregate / augment more specific Layers Contexts instances. Layers Context types aggregation signatures.
* Domain services dataflow: Contexts instances invoked according domain / range signatures. Events: LayerType::onMessage (matches / next layer). Traversal aggregates matchings from previous layers until aggregated Contexts.
* Context: Layer type / instance matching selector Context. Domain Services invoked passing Message as argument:
* Object : yields Aggregation statements. Contexts / Occurrences domain / range.
* Sign : yields Activation statements. Occurrences / Kinds domain / range.
* Value : yields Alignment statements. Kinds / Resources domain / range.
* Domain Service behavior: Context instance URI APIs. IO: Message. Domain features / busines logic (express declaratively: TBD). Process invocation result Message(s) recursively (dataflow). Hierarchy Context matching: reified URIs / Contexts.

**Topics:**

Mision

Vision

Objectives

Problem Description

Solution Approach

Features

Backends: RDF(S), OWL, Property Graphs, SPARQL, ShEx, SHACL. Native inference (upper ontologies).

Backends introduction: OGM, features (Service Resources).

Quads introduction.

Encoding / Addressing / IDs.

Normalization.

Parsing / Protocols / Messages.

Features in practice. Use Case: "Goals App".

Ontology matching / alignment: Merge domains / backends data, schema and behavior (reified as schema / dataflows).

Message based / Event sourcing driven core (OntResource Dataflow integrations).

Message connectors / translation interfaces (via Service Resources):

Resource oriented REST HATEOAS / HAL Hypermedia facade. Endpoints, encoding, Forms, Flows.

Connectors examples: Apache Metamodel, JBoss Teiid, Apache Any23, R2QL, etc.

JDBC / JCA: Drivers / Resource Adapters.

Declarative Application Stack (Hypermedia example): XML, XSL, XPath, XQuery, XLink, XPointer, XForms, XUL).

Models:

OntResource:

Abstract layer: Alignment APIs. Template methods. Hierarchy key / value reification (alignments). Meta Meta Model.

Role : Occurrence : Object : OntResource;

Meta Resource (DOM): Role.

Service Resource (I/O, etc): i.e.: JDBCOntResource.

OntResource: reactive event driven processor (topics / queues: streams of corresponding signatures (Kinds) according specific OntResource Role subclass.

Meta Models. Facets. APIs (layers templates):

Augmentation:

(Relation, Kind, Statement, Role);

(Relation, Statement, Kind, Resource);

Dataflow:

(Augmentation, Template / Predicate, Mapping, Transform / Resource);

Dimensional:

(Dimension, Unit, Measure, Value);

APIs interact with each other when Roles in different Facets refers to the same OntResource.

Augmentation:

Order.

Activation: Dataflow fires / consumes stream events (Kinds / Roles). Concrete Role hierarchy instances / OntResource templates

Aggregation: Functors. Role layer instances involved in Activation matching.

Alignment: Transforms. Involved matched Functors mapping application streams.

Model Levels: Backend, Domain, Session DOM Roles resources aggregation.

FCA. Lattices:

Meta Model(s) Facets layers as axes of an FCA context (object / attributes). Axes use FCA scaling for grouping of Roles instances. Concepts, attributes, values:

(type / val) x (type / val): (type / val);

Reify contexts SPO as new FCA context axes: aggregate metadata.

Sorting, matching, augmentations, query, browse, ontology completion (inferences).

Learning: ML embeddings encoding scaled (type / val).

TBD: mcls, cls, occur, inst template roles.

TBD: Sets. IDs. Hashing (bitstring), layout rules, set encodings.

TBD: Ontology matching: Triadic FCA Context (object, attribute, condition). Fuzzy / rough sets (papers).

TBD: Protocol: Endpoints SPI / API implementation. Uniform Hypermedia interface. Messaging layer (levels). JDBC / JAF / JMS / JCA Connectors / Adapters.

**TOC:**

**Distributed Integration and Consistency for Knowledge Semantics**

[Introduction 1](#_gjdgxs)

[Metamodel 3](#_30j0zll)

[Layers 5](#_1fob9te)

[Aggregation 8](#_3znysh7)

[Augmentation 9](#_2et92p0)

[Functors 9](#_tyjcwt)

[Messages 10](#_3dy6vkm)

[Dataflow / Activation 11](#_1t3h5sf)

[Services 12](#_4d34og8)

[Encoding 13](#_2s8eyo1)

[RDF Quads 15](#_17dp8vu)

[Semiotic Encoding 18](#_3rdcrjn)

[Normalization 20](#_26in1rg)

[Parsing 21](#_lnxbz9)

[Protocol 22](#_35nkun2)

[Addressing 23](#_1ksv4uv)

[Low level 23](#_44sinio)

[Hypermedia Activation 25](#_2jxsxqh)

[Facets / Levels 26](#_z337ya)

[Features / APIs 27](#_3j2qqm3)

[Features 29](#_1y810tw)

[Domains schema 29](#_4i7ojhp)

[Ontology matching 29](#_2xcytpi)

[Augmentations 32](#_1ci93xb)

[Deployment Use Case: Goals Application 32](#_3whwml4)

# **Introduction**

Distributed Integration and Consistency for Knowledge Semantics.

Property graphs allow properties (key/value pairs) to be associated with both nodes and links in directed graphs. This allows you to annotate links with information such as the start and stop times for when the link is valid, its provenance, a statement about its quality and so forth.

Property / labeled graphs. Dot notation / encodings. Nodes / arcs links / attributes: metaclass / class / instance / occurrence levels. Individual node / arc label or other node / arc in rel context (subjects in context occurrence attribute / value data model). Links / rels subjects (with labels).

Signatures: Context / Subject, Occurrence, Attribute, Value. Metamodel: metaclass, class, instance, occurrence upper ontology (primitive) property graphs categories / types.

Functional. Sets, groups, categories. ADTs. Metamodel. Augmentations / Functors. Layers, Aggregation, Alignment, Activation (type) categories / transforms (encodings).

ERP Ontology: from concepts to application gestures / purposes (high level APIs).

Description

Distributed Knowledge Base for Functional Syndicated Application Integration and Virtualization Framework. Plug existing backends (applications / datasources / services) via in an EAI / ESB fashion. Provide semantic augmentation of learned applications metadata (data / schema / behavior) via Augmentations (Aggregation, Alignment, Activation).

Distributed systems / micro services access to shared data. Shared data consistency / inference / MDM. Ontology matching. Integration features: rules and flows based services composition.

Distributed P2P (Blockchain) approach of data synchronization between peers for ease of deployment patterns election and datasources integration (APIs, microservices, etc.).

Augmentations:

Aggregation: infer roles in contexts (see Metamodel Layers). Regression: Person class in Employment interaction referring context, Developer role.

Alignment: infer links / attributes. Clustering: from multiple occurrences of same entity in diverse data sources, complete missing / new information.

Activation: type inference. Classification: determine metaclass / class / instance / occurrence subject roles for corresponding entity attributes and values.  
  
Hypermedia APIs for reactive integration of addressable resources representations. Content type driven semantic augmentation / annotations. Plugged backends synchronization. Data, information, knowledge levels of interaction contexts abstractions (from dimensional to plain facts).  
  
Domain Driven Development. Use cases "problem spaces" translation of aggregated domains services into underlying resources. Domains Use cases abstractions (problem "spaces" / ontologies) enabling domain translation / exchanges / integration. Declarative Application Design.

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.

# **Metamodel**

Layers are implemented as an RDF Quads hierarchy aggregating each one on top of another. The idea is that aggregating 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.

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.

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.

Link grammar: (employer, employee). Categories, monads. Reference Model.

Link Grammars. Types: links left / right types defined when a shape / slot match satisfaction occurrs (roles).

Parsing: extract propositions, knowledge assertions (in a domain ontology Behaviors). Assert propositions links, order, concepts relations (between domains). Link Grammar. ISO TMDM / TMRM.

Parsing: extract prescriptions, knowledge rules (in domain ontology concepts relations: causal, requirements, etc.). StratML.

Parsing: infer possible statements propositions / prescriptions productions. Link Grammar Disjuncts. Embeddings.

Message: Functor Declaration. (events / grammar: protocols).

Augmentation: Functor Instance.

Context hierarchies: categories of wrapped subject occurrences. Statement Context monad (category) for Subject occurrences.

Context category instance identity: Employee Kind, Work Behavior, etc. Context statement attribute / value aggregation. Key / value abstraction scoped in category type / instance.

Functors: Layers (Context categories) aggregation / transforms. Augmentations.

Augmentations: Categories Aggregation (types), Alignment (contexts), Activation (roles / matching in interactions).

Locators (keys): (metaclass, class, instance, occurrence);

Metaclass: Transform OntResource

Class: Message Augmentation

Instance: Statement Role

Occurrence: Flow Kind

Locators (keys): local / remote keys. Navigation. Paths. Legends. Signatures (streams).

Monadic (locators / signature streams based) category wrappers Functors:

M<T>::flatMap(functor(T) : U) : M<U>;

Dado rango y alcance, universo: U de una relación R, inferir dominio y codominio, campo: C.

From Object (O) extension /instances to Context (C) intension / class.

Aggregation (types) functor signatures: stream.

Alignment (contexts) functor signatures: stream.

Activation (roles / matching in interactions) functor signatures: stream.

Encoding: IDs. Embed metaclass, class, instance, occurrence metadata (context, role, attributes, values). Functional APIs. Wrappers / Transforms (augment: aggregate / classify, roles, properties "graph" rels). Polygon Vector Space Model. ANN embeddings / autoencoders.

Forms / Flows Dialogs / Contexts. Protocol. Resources, addressing, representations, navigation / traversal: properties "graph" rels (Wrappers / Transforms). Functional APIs.

Sets encoding: properties in axes (kinds). SortedSet (hierarchies). Metaclass, class, instance, occurrence properties in axes for CSPO IDs. Augmentations: property graph rels navigation / traversal. Dialog Forms / Flows "state" contexts.

Contexts Wrappers kinds Transforms / Traversals functors: Augmentations declaratively stated in upper Context layers (kind classification, kind roles, kind attributes / values).

Dialog Forms / Flows "state" Contexts browsing (upper Context SPO kinds: current context streams).

Augmentation navigation of Transforms / Traversals as a Context (streams / filters). Levels / reification.

Order. Iteration. Predicates (resource meta / domain / kinds). Streams filter, conditionals, jumps. Aggregation. Functional mapping / reduce, etc.

Resource API (layer roles):

Statement: Resource (CSPO: Statement) Property graph URL wrapper. URL occurrences aggregate. Functional occurrences properties (roles / streams) for Statement wrapped URL APi:

Context: Metaclass URL occurrences.

Subject: Class URL occurrences.

Predicate: Instance / Occurrence URL occurrences.

Object: Occurrence / Instance URL occurrences.

Statement wrapped URL occurrences functional roles Kinds API: Map<Role, Kind> (reified in Metamodel) in occurring Kind query / stream context selectors: ID (URL), layer type, layer role, layer kind occurrences context selectors.

CSPO Kinds (streams functors) declaratively stated in aggregated layers CSPO occurrences (kind classification, kind roles, kind attributes / values).

Augmentation navigation of kinds Transforms / Traversals as streams / roles (filters). Levels / reification: kinds from Statement / roles layers (reification / levels axes).

Query / stream context selectors: ID (URL), layer context type, layer context role, layers kinds occurrences. Transforms (functor kinds: augment / browse query context according kind specification with corresponding statements).

Levels: Grammars (kinds functors dataflow signatures). Productions: Augmentations (parsed / produced in navigation contexts). Dataflow order (sets / hierarchies).

Monad: context type. Metaclass.

Monad: wrapped type. Class.

Monad: wrapped value. Instance.

Monad: wrappers hierarchy context type instance. Occurrence.

Unit. Type constructor (hierarchy context / member types / values factories). flatMap / map / flatMapN.

Monad: context type case classes hierarchy? Factory methods, hierarchy types / members values / signatures (wrapped context / values) cases (predicates):

Optional: None | T [case context | case context | case value signature]

Result: Error | OK [case context | case context | case value signature]

Writer: Value, List<S> trace.

Dataflow:

Input statements: Augment. Aggregate, build layers contexts representation quads. Parse:

Monadic parser combinator: aggregated metamodel nested layers contexts corresponding wrapper / wrapper hierarchy types DOM / AST (from contexts quads aggregation nesting hierarchy levels).

Augment: signatures dataflow. Apply input statements: new inputs / kind new attributes (person, employee -> position, salary). Parse.

Protocol: Forms / Flows augmented / parsed representation / metamodel I/O. HAL / HATEOAS endpoint encoding for navigation, transforms and inputs augmentation.

Protocol: Forms / Flows Dialog / Prompts resolution. Context roles, wrapper kinds navigation / transforms declaratively stated in encoded representations.

Actor / Role. Dynamic Object Model (OGM / Kinds). Golden Braid: Metaclass, class, instance, occurrence relation relative to layer context levels.

Reference Model:

Type : Value;

(Value, Type);

(Type, Type);

(Value, Value);

Aggregation: cons cells tree traversal. Layers contexts representation. Dataflow augments updates / append corresponding tree cells: (first: (rest: nil). (C (S (P (O))));

Aggregate / Augment inputs / transforms: Parsed model streams. Reactive data structures.

Encoding: addressing, semantic graph networks. URN overlay semantic addresses encoded mappings.

Functor: mapping between categories A / B using a function. Map, alcance, rango, dominio, imagen (infer, aggregate). Connections: same number of items (inyective, biyective). Function: de wrapper en wrapper (morphism new image). Message (functor) / Augmentation (transforms) Metamodel reifications.

## ***Layers***

Productions:

Inputs / Outputs: aggregate Augmentation / Message from input semiotic alignments. Protocol.

Inputs / Outputs: HATEOAS protocol browse / transform aggregated Context (model) Messages / Augmentations.

Inputs / Outputs: Layers hierarchy Augmentation / Message population / transforms. Meta Resources (Message / Augmentation reified context recursive schema / transforms declarations) I/O.

Monads: Quads Contexts wrapper of Occurrence / Subject aggregations. Root type: Resource. Kind (model / semiotic / domain meta resources) functors filter / traversal (streams: flatMap Resource set specifications).

Wrapped resources holds references to its wrapping occurrence contexts (Resource root type interface, contexts hierarchy / reification levels: upper layer instances reify layer occurrences in contexts). Context siblings, parent / child hierarchies kinds / attribute / values browse / filter / streams (axes: type activation, roles in context, attributes / values augmentations).

Wrapping unit / bind: layers traversal until container wrapped category type is met. Example: Behavior(s) of an Entity. Augmentations (types, roles, attributes reified model / domains aggregations). Example: Model, SK, P; Domain, someSubjKind, somePredicate.

Form / Flow: state navigation / browse. Dialogs / Prompts (contexts). Messages I/O: Augmentation signatures (kinds) streams.

Monads: Messages. Augmentations according signatures. Broadcast. Key / Value, Event sourcing routes. ServiceMix / JXTA: local Peer OSGi bundle. Camel ActiveMQ routes through local peer (jxta://localhost).

Messages: Augmentation according signatures. Broadcast. Key / Value, Event sourcing routes (levels). Form / Flow Dialogs / Prompts Messages for signatures (kinds) of unknown Resources resolution (layers wrapping traversal, CSPOs layers resolution for CSPO signatures / attributes / values).

Layers hierarchy:

(Context, Object, Concept / Sign, Value);

Semiotic roles. Kinds, grammars, levels. Contexts aggregations.

Resource: Plain inputs.

(Resource, Resource, Resource, Resource);

Role / Kind: Aggregate input statements CSPO roles (S) and their corresponding Attributes / Values.

(Role, Resource, Resource / Attribute, Resource / Value);

Statement: Aggregate Resource occurrences in CSPO Role(s) for Attributes (Kinds).

(Statement, Role, Resource, Resource / Attribute);

Entity: Semiotic Sign (Concept) Value (OntResource). Aggregate equivalent (ontology matching Resource extension for Entity intension.

(Entity, Statement, Role, Resource);

Template: Grammar kind. Semiotic Concept / Sign. Layer instance "template" kind (transform). Object occurrence. Aggregate Entities with similar kinds (properties). Occurrence Role / Kind.

(Template, Entity, Statement, Role);

Mapping (Instance) : Semiotic Object. Layer category type instance. Statements which are results of Behavior Flow (Statement: extension, Mapping: intension). Aggregate set of statements regarding the same concept (Mapping) in different occurrences.

(Mapping, Template, Entity, Statement);

Flow: Event declaration (I/O). Entities which are results of Behavior Flow (Entity: extension, Flow: intension. Mappings objects in Template Kind roles). Semiotic context.

(Flow / Augmentation, Mapping, Template, Entity);

Behavior: Message Specification (I/O).

(Behavior / Message, Flow, Mapping, Template);

(Measure, Behavior, Flow, Mapping);

(Unit, Measure, Behavior, Flow);

(Dimension / Axis, Unit, Measure, Behavior);

(Value, Dimension, Unit, Measure);

(Concept, Value, Dimension, Unit);

(Object, Concept, Value, Dimension);

(Context, Object, Concept, Value);

Inputs feedback. Semiotic encoding (feedback / grammars / levels).

Meta Resources (Message / Augmentation reified schema / transforms aggregation / recursive declarations).

Aggregation: materialize grammar statements of contexts recursively. Activation (types). Alignment (augmentations).

(Aggregation, Message, Augmentation, Mapping);

(Alignment, Aggregation, Message, Augmentation);

(Activation, Alignment, Aggregation, Message);

Encoding. Model layers:

URIResource context: CSPO form. RESTful / HAL monad: HTTP category functors.

Inputs / Outputs. Semiotic encoding (feedback / grammars / aggregated levels):

(Context, Object, Concept, Sign / Value); Semiotic roles (object as context / value, value as object, etc.) Labeled property graph / lattice. Inference rules (transforms) over roles. Kinds, grammars, levels.

OntResource context: (Resource, Occurrence, Attribute, Value) form. Aligned (matched) URIResource(s).

Resource context: OntResource (aligned / matched URIResource) occurrences in reified Role in Statement.

(URIResource, URIResource, URIResource, URIResource);

(OntResource, URIResource, URIResource, URIResource);

Aggregated URIResource OntResource attributes / values (recursion to attributes / values OntResource).

(Transform, OntResource, URIResource, URIResource);

Augmentation (production) Resource. Semiotic Sign / Value.

(Template, Transform, OntResource, URIResource);

Augmentation (production) grammar kind. Semiotic Concept.

(Mapping, Template, Transform, OntResource);

Semiotic Object.

(Augmentation, Mapping, Template, Transform);

Production (grammars). Semiotic Context.

(Message, Augmentation, Mapping, Template);

Grammar rules. Functors.

(Context, Message, Augmentation, Mapping);

Model.

(Resource, Context, Message, Augmentation);

(Role, Resource, Context, Message);

Reified CSPO / Resource, Occurrence, Attribute, Value Resource role types in Resource occurrence / context.

(Statement, Role, Resource, Context);

(Entity, Statement, Role, Resource);

Aggregated "subject" occurrences of Resource in Role in Statement(s).

(Class, Entity, Statement, Role);

Aggregated Entity Role occurrences type (attributes).

(Kind, Class, Entity, Statement);

Aggregated kinds / roles ("interfaces") of Class occurrences.

(Flow, Kind, Class, Entity);

Action "instance". Entity of Class performs role (Kind) of Behavior Flow.

(Behavior, Flow, Class, Kind);

Action "class". Statements: propositions, prescriptions, rules, productions. DCI / Link Grammar. Context satisfaction (rules).

(Measure, Behavior, Flow, Class);

(Unit, Measure, Behavior, Flow);

(Dimension, Unit, Measure, Behavior);

(Value, Dimension, Unit, Measure);

(Concept, Value, Dimension, Unit);

(Object, Concept, Value, Dimension);

(Context, Object, Concept, Value);

Inputs feedback. Semiotic encoding (feedback / grammars / levels).

Statements: propositional, rules, reasoning. Reification. Value: OntResource (instance); Concept: class, Object: metaclass; Context: occurrence.

Order. Order: Grammars kinds sets / hierarchy (signatures). Common super type / kind / role / occurrences. SortedSet.

Flat map functor: Kind (non terminal) instance to be extracted from (Augmentation) context (matching rules) according Message specification inside corresponding wrappers until context category. Grammar (rules signatures, messages).

## ***Aggregation***

Super Kind / sub Kind hierarchy relationship is given by a set of Kind Attributes being super set / sub set of each other. SortedSet (order, iterations).

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).

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

Meta Model: Encode / reify Model(s) w./ Meta Resources and Model Context(s) hierarchies. Meta Model: Encode order, iteration, conditional flow. Dataflow.

Encoding: Kind hierarchies / Grammars

(CK, SK, PK, OK);

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.

Graph Execution Semantics: Dataflow by Context Kind domain (Subject Kind) / range (Object Kind) matching Forms / Flows. Ontology Matching. Upper ontologies. Primitives.

## ***Augmentation***

Functor application.

Resource Monad:

Encode protocol functors. Endpoint address activation behavior facades. Graph state / rels traversal: Monad encodes entire state location flows to current CSPO URL IDs state (traceability in interaction context rels). Abstract Form / Flow attrs / rels.

Functors resolution on API addresses URLs: resource monads rels / attrs activation.

Behavior layer renders domains possible aggregated augmentations / messages of model functors composition. Rendered in domain levels as concrete contexts operations: named context operations over abstract model functors behaviors.

Augmentations:

Aggregation: infer roles in contexts (see Metamodel Layers). Regression: Person class in Employment interaction referring context, Developer role.

Alignment: infer links / attributes. Clustering: from multiple occurrences of same entity in diverse data sources, complete missing / new information.

Activation: type inference. Classification: determine metaclass / class / instance / occurrence subject roles for corresponding entity attributes and values.

## ***Functors***

Functor: mapping between categories A / B using a function. Map, alcance, rango, dominio, imagen (infer, aggregate). Connections: same number of items (inyective, biyective). Function: de wrapper en wrapper (morphism new image). Message (functor) / Augmentation (transforms) Metamodel reifications.

Flat map functor: Kind (non terminal) instance to be extracted from (Augmentation) context (matching rules) according Message specification inside corresponding wrappers until context category. Grammar (rules signatures, messages).

Transform (Resource).

Mapping (Occurrence).

Template (Grammar Kind).

Augmentation (Production).

Message (Rule / Functor).

Core rules: Augmentations. Reified CSPO roles (kinds, etc. meta resources) terminals / non terminals.

Domain rules: instance CSPO roles (grammar) terminals / non terminals.

## ***Messages***

Functor declaration. Resource set specification.

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

Order: Common super type / kind / role / occurrences. SortedSet.

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

Signature based bus dispatch / application.

Signatures

CSPO Context Kind (Statement Subject Kind + Object Kind). Context Dataflow domain / range (Context as reactive streams producer / consumer).

Routes / Dataflow

Routes: Dataflow pub / sub bindings between matching signatures.

Core Model and Domain driven Message flow layout (Mappings).

Event Bus / Messages / Addressing

Dispatch Event into Dataflow Route.

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:

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.

## ***Dataflow / Activation***

Levels: Grammars (kinds functors dataflow signatures). Productions: Augmentations (parsed / produced in navigation contexts). Dataflow order (sets / hierarchies).

Event I/O (Quad Statements). Augmentations application. Dispatch: Signatures (Kinds). Apply transform (materialize event). Emit events (endpoints / filters / criteria streams signatures). Update graph listening events according its contents dataflow.

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.

Dataflow:

Events: Message (functor) declarations matching Augmentation Mappings by signatures.

Basic hypermedia browse (CRUD / transforms). HTTP verbs bound Message functors compatible for Resources (REST). Domain aggregated functors application (signatures).

Encoding: XML / XSL / Template Scripts (functional runat: peer dialogs / reactive callbacks). Mappings declarations / encodings (primitives, wildcards, variables, placeholders templates: actual / result of, possible).

Formalization: Functional / Object APIs. Reference / Data model. Sets, categories, models. SortedSet (hierarchical structures).

Semantic resolution:

Query Resource(s) satisfying “criteria” (i.e.: Object(s) for predicate / Augmentation Mappings Forms / Flows) IDs by IDs resolution pattern: (Message applicable signatures : resolution result: Transform).

Query Resources by role in context.

Query Resources by attributes / values.

Query Resources by identity / type.

Streams. Signature filters:

Streams: URIs, Resource, Statement, CSPO Roles, Kinds. Dataflow: index / order signatures dispatch, reactive (signatures).

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).

Messages: Dataflow Template matches signatures (Session level, enrichs Message with Model / Dialog prompts / contents alignments). Augmentation Functor applied over Message contents (Interaction level). Transform matching output signature emits (Session level, populated / prompts) output Message (parse Transform).

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).

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.

Messages: Service Context URIs: Signature for face recognition (image URI / resource : domain, detection / search results endpoint / placeholder : range). Others services: ML Classification, Clustering, Regression, Services Index, Naming, Registry. Presets "inferred" models and augmentation services (populated / online learning).

Augmented Semantic Content Types (img/xml;facesCoords).

Upper Ontologies. Load. Grammar level services (schema browse, possible flows query / browse). Message: wildcards, variables, placeholders.

## ***Services***

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

External services I/O modelled as Resources in layers contexts. Layers (session, dialog, etc.). Node, Peer, Client, Connector, etc. Reactive / Event Driven. REST HATEOAS (dataflow).

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.

Index, Naming, Registry.

MapReduce.

Blockchain DIDs.

MapReduce (Encoding).

Backend Connectors.

DCI / MVC: JDBC / OGM / ORM / JCA / Activation JAF / Process Flows (state) semantics.

Declarative hypermedia: REST / HAL / HATEOAS functional protocol.

CMS / Wiki (API / Protocol / DAV). Docs. Forms (Docs APIs)

# **Encoding**

Encoding:

Encoding: IDs. Embed metaclass, class, instance, occurrence metadata (context, role, attributes, values). Functional APIs. Wrappers / Transforms (augment: aggregate / classify, roles, properties "graph" rels). Polygon Vector Space Model. ANN embeddings / autoencoders.

Forms / Flows Dialogs / Contexts. Protocol. Resources, addressing, representations, navigation / traversal: properties "graph" rels (Wrappers / Transforms). Functional APIs.

Sets encoding: properties in axes (kinds). SortedSet (hierarchies). Metaclass, class, instance, occurrence properties in axes for CSPO IDs. Augmentations: property graph rels navigation / traversal. Dialog Forms / Flows "state" contexts.

Contexts Wrappers kinds Transforms / Traversals functors: Augmentations declaratively stated in upper Context layers (kind classification, kind roles, kind attributes / values).

Dialog Forms / Flows "state" Contexts browsing (upper Context SPO kinds: current context streams).

Augmentation navigation of Transforms / Traversals as a Context (streams / filters). Levels / reification.

Order. Iteration. Predicates (resource meta / domain / kinds). Streams filter, conditionals, jumps. Aggregation. Functional mapping / reduce, etc.

Encoding: metaclass, class, instance, occurrence (contextual / nested / orders / ops) CSPO IDs. CURIEs.

Encoding: Sets CSPO Contexts specification (sets quad encoding).

Encoding: Functor application. Predicate: functor behavior, domain: statement predicate, transform / range: statement object.

Encoding: Levels (OntResource context hierarchy) reification: Message as Predicate, etc. Resource Monad (context statement / signatures). Functor aggregation: levels (type, role, alignment).

Encoding: Grammars. OntResource hierarchy reification: rules (contexts) / non terminals (reified Predicates / Kinds). Aligned OntResource URLs: terminals. Augmentations: productions (functors).

Link Grammars. Types: links left / right types defined when a shape / slot match satisfaction occurrs (roles).

JAF. Index / Naming / Registry (HATEOAS Forms / Flows navigation / states): DCI / MVC Engine.

Dimensional:

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

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

Assert knowledge: 1h -> 60min;

Assert: dom-lun-mar-mie-jue-vie-sab;

Assert: 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.

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. SortedSet hierarchies.

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

(Value, Previous, Distance, Next);

Person, Single, Marriage, Married;

Man, Single, Marriage, Husband;

Woman, Single, Marriage, Wife.

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?).

Encoding:

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. SortedSet hierarchies (3 digit octal set membership values).

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.

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

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

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

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

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

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

Model: Contexts class subjects instances occurrence role kinds attributes / values.

Quad encoding: Context relative IDs (polygon). 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.

Dataflow: Order, Forms, Flows (Signatures, 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.

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

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.

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

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.

## ***RDF Quads***

RDF Quads / Object Mapping (DOM / OGM)

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).

RDF CSPO Quads store backend. RDFS / OWL inference

Backend:

Messages: Events IO / Persistence: Saga Activation / Passivation populating Node local Quad store / persisting peers via DIDs (inference enabled distributed consistency) semantic (resolvable / discoverable) identifiers.

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.

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

Quads Context / Object: class by intension / extension. Transform matches Context signature, filters by Object(s) extension. Resource(s) specification.

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

Deployment / Use Cases:

Purpose driven hypermedia activation Augmented Semantic Content type activation. Messages / gestures. Rules (commands / verbs).

Browser referring context (Work, Peter, Employee). 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

Core (upper / onto) Messages: Getters, setters, nav, etc. Domain Messages: raiseSal: setSal(sal \* increment); promotion: setPosition.

Event sourcing / tracking: married -> marriage occurred. Dataflow: Messages hierarchy. Aggregate contexts from coarse to fine grained  transforms (raiseSal -> setAttr, single - marryWith).

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. Execution graph.

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

Processor which acts upon Resource events. Materialize results. Specify declaratively augmentations by means of messages. Metamodel model events I/O, flows.

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.

Metamodel Message event driven I/O protocol:

Discovery: All model kinds are browseable / discoverable. Encode behavior in statements / graph: Comparisons, order. Sort. Order (kinds hierarchy?) Pattern matching, iteration, jumps. Discovery: routes / signatures, next event in bus / graph. Dataflow.

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

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 (SortedSet).

Dimensional input set model aggregation 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).

I/O: Augmentations. Basic operation. Resource Set Specification (SortedSet / Statement) matching Model which returns augmented Message response (Model I/O).

## ***Semiotic Encoding***

Subjects: attributes / values. Occurrences: contexts / roles.

(Context, Occurrence, Attribute, Value);

(Context, Sign, Concept, Object);

Metaclass, class, instance, occurrence.

Assert order / hierarchies / relations in dimensional axes. Containment (sets).

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). Compose IDs (hierarchical graph properties encoded string) from outer to inner resources (Context, Kind, Occurrence, Role, Resource). "Operable" IDs (ClassIDs / InstanceIDs: Meta Model reifications / occurrences).

Message parsing (Template, Message Context) matches Form / Flow Augmentation Mappings signatures.

(Template, Context, Attribute, Value); Value as Context: hierarchical models. Same attributes: types / collections.

Semiotic (encodings):

(Context, Sign, Concept, Object);

Object as Sign: Object properties (Concepts).

Semiotic encoding: encode Meta Model (syntax facet) / Functional (Meta Model encoded pragmatic facet) / Dimensional (Meta Model encoded semantic facet) data / reference model (by contexts / upper alignments / aggregation).

Facets contexts semiotic encoding. Facets IO (events) by semiotic encoding of facets input layers.

(Context, Sign, Concept, Object);

Roles encoding: Object as Sign (properties), Sign as Object (types / roles), Concept as Sign / Object / Context, etc.

Grammars encoding: express models contexts layers and augmentation templates in input contexts.

DCI / DOM: Subject, Context, Occurrences, Roles, Attributes, Values, Activation.

Semiotic, Meta Model, Dimensional, Functional DOM. DOM Contexts. DOM Functors. Resources Contexts (CSPO Monads: encoding / addressing).

Activate able DOM Contexts: DOM views (object mappings, etc.).

Map Reduce encoding inputs, grammar templates context mappings. Emit Semiotic reference / data model parent / child properties encoding (Context / SPOs, Context:Subject / POs, etc.).

Model / Semiotic reification.

Facets / Levels / Shapes: Aligned entities / values occurring in aligned models / dimensions.

(Context, Sign, Concept, Object);

Resource Monad / Message Functor (Contexts hierarchy parent). Resource<Context>. Events bindings. Message / Augmentation event declarations / instances.

Order: SortedSet, hierarchies, common upper types rels.

## ***Normalization***

Radix, Digit, Position. SPO: radix:digit. Radix: resource count. Digit: resource radix index. Position: context hierarchy aggregation. Normalized radix encoding: metaclass, class, instance, occurrence context metadata operable IDs. Example: Base 2 normalization, concepts lattice bitstring (routing tables), boolean operations metadata extraction / traversal (contexts masks). IPv6 addresses. Masks (contexts), routes. Address resolution resolves recursively expanded (CSPO hierarchy contexts / occurrences / attributes / values) resources representations (graph statements members nested browseable addresses of ID contexts). Addresses representing behaviors (HATEOAS: browse representation links / addresses renders state / operations, i.e.: address plus according referrer context and representation addresses links states CRUD operations).

Normalization: contextual form IDs. Internal four sides polygon angles for each CSPO addressable IDs. Embed operable meta data in CSPO contextual IDs. Vector space model like. Polynomial coefficients / factors. Adjacency matrix / tensor. Embeddings.

CAM (Ternary bitstring). Routes. CSPO Functional Mappings (contextual / occurrences vector IDs).

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). Compose IDs (hierarchical graph properties encoded string) from outer to inner resources (Context, Kind, Occurrence, Role, Resource). "Operable" IDs (ClassIDs / InstanceIDs: Meta Model reifications / occurrences).

Encode common upper Semiotic / Dimensional Model: Reference Model.

Encode Kind / Context hierarchies.

Encode Augmentation(s) as Resource descriptions (Resource occurrence of Augmentation).

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

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

Object occurrence of Predicate.

Sets. Quads. SortedSet.

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).

Encoding: recursive resource quads encoding hierarchy, order, class, instance, attributes. Operate inferences over (upper) patterns (bitstring / lattice). Meta Model, Facets, Levels. Specifications: Signatures, Forms, Flows (encode events / transforms provenance).

Order / Comparison: Ternary truth values results (previous / parent, current / OK, next / children) according Predicates for a Message resource set and and a Mapping comparison (apply augmentation). Octal relations.

## ***Parsing***

Parsing: extract propositions, knowledge assertions (in a domain ontology). Assert propositions links, order, concepts relations (between domains). Link Grammar. ISO TMDM / TMRM.

Parsing: extract prescriptions, knowledge rules (in domain ontology concepts relations: causal, requirements, etc.). StratML.

Parsing: infer possible statements propositions / prescriptions productions. Link Grammar Disjuncts. Embeddings.

Functor / Parser signature:

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

Behavior statements.

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).

# **Protocol**

Forms, Flows: Meta Model Behavior. Forms / Flows APIs: grammar / semiotic / levels / dimensionally augmented / aware.

## ***Addressing***

Addresses browse state transforms: navigation renders context layers statements transforms with contextual browsing state (IDs, referrer, contextual / occurrences vector IDs, metadata) as parameters. Contexts navigation (Forms, Flows) as functors. Navigation state transforms: possible activations context / argument / attributes / values. Dialog / prompts (argument resources navigation state transforms).

Encoding: nested shapes of recursive cuads (till primitives) identifiers. Patterns / expressions: wildcards, variables, placeholders:

[[123, 456, \_b, $a][\_b][\*][$a]]

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

Siblings: previous / next (Semiotic containment relationship / roles). To do.

Hierarchies: common parent / children (Semiotic containment relationship / roles). To do.

Semiotic model: reification / primitives / upper ontology (meta model, inferred / aggregated). Levels: syntax / grammars, semantics, pragmatics. To do.

Augmentations: addressing (HATEOAS) browsing of: Aggregation, Alignment, Activation transforms (Behavior Messages).

Message: Augmentation (materialised Resource set) specification (functor). Event declaration (I/O patterns bindings). Context layers monad publishes / subscribes (dataflow bindings pipelines).

Ontology matching: signatures encoding. Dimensional ordered aggregated measures. Sets. Semiotic reification.

## ***Low level***

Forms, Flows: Meta Model Behavior / Flow APIs grammar / semiotic / levels / dimensionally augmented / aware. From core model levels to dialog / gestures interaction relations / resources (session level model resources).

Model: RDF Backend.

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

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.

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): Basic type inference. Applied over layers CSPO during Activation Augmentation.

Source / Session / Pragma levels. DCI. Data / Information / Knowledge. Syntax, Semantic, Pragmatic. Model state: Context (Resource : data), Kind (Grammar : schema), Dimension (behavior). Context Kind(s) signatures: Dataflow.

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). CQRS. Dialog (EAI pattern: Isis docs).

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 statement CSPO roles.

Alignment: Infer layer missing / deducible attributes and values for CSPO Subjects.

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

Ontology matching: Dataflow: sort statements. Form / Flow Dimensional Metadata. 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.

## ***Hypermedia Activation***

Addressing (hypermedia):

Content Type: Metaclass

Resource: Class

Representation: Instance

Address: Occurrence / Response (Materialised Message Augmentation)

Request: Message (state flow). Context DOM event API: Request Message Resource in possible domain / range / mapping contexts.

Purpose / content type driven (state / rendering / roles / links /contexts / data / flows / attributes) declarative hypermedia activation application. Model, Application, Domain ontology / upper resources (connectors). Generic API / metamodel (DCI: Form / Flow) client. Extension protocols / APIs. Meta Model levels: from abstract to concrete applications.

Encoding:

Rendering: S: current document URL, P: link tag body, O: href, rel: Context (referrer). Navigation: GET / headers. GET (navigate, possible resources, posible contexts / subjects / attributes): CRUD / Contexts aggregation / transforms / matching. Encoding: CRUD / browse layers (CSPO Patterns Forms / Flows layers de-aggregations / faceted traversals).

Encoding, APIs: REST HATEOAS, JSON-LD, HAL. Distributed (normalized) address ID spaces.

Layers down / up traversal:

C: Anchor rel (referrer);

S: Current URL;

P: Anchor tag body;

O: Anchor href;

Layers up / down traversal:

C: Current URL;

S: Anchor rel / referrer;

P: Anchor tag body;

O: Anchor href;

Activation Protocol:

Protocol: Input statements for querying augmented knowledge (Specification Forms / Flows). 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.

Dataflow embedding: Resources reifying global state. Specifications: Forms, Flows. Augmentation Dataflow: 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.

Source / Grammar / Pragma Levels.

Functional / Dimensional / Semantic Facets.

Reactive Entities: Resource, Model, Message, Kind.

Entities: ID (routes), State (ctx / rel pointers, occurrences). Streams, Dataflow (routes / bindings: addressing).

Transforms, Augmentation (functors / mappings).

Dataflow: Message / Model / Augmentation / Model / Message.

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.

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).

## ***Facets / Levels***

Forms, Flows: Meta Model Behavior / Flow APIs grammar / semiotic / levels / dimensionally augmented. From core model levels to dialog / gestures interaction relations / resources (session level model resources).

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 / Mapping Transform).

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.

Levels: (upper) ontology aggregation layers from primitives until application / domain concepts.

Model source level: 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 (domain) 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).

Reification: All Model layers and types reified into Resource hierarchy (sub class relationship).

Example: Source, Grammar, Pragma (Session Dialogs / Prompts) levels.

## ***Features / APIs***

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.

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

MDM. Provenance. Versioning. Dimensional context values / queries. Model Facets APIs: Functional, Semiotic, Dimensional Dataflow contexts / order / roles HATEOAS APIs workflows. Protocol: Dialog. Browse / analyze / transform "activations" (REST / JAF) according Facets. Clients / Connectors. OGM.

Contents / Features (Mision / Vision). Distributed consistent Knowledge Applications. Trust. Consistency. Event sourcing. Inferencing (of distributed state). Reconciliation.

Certify distributed Entity / Subject Identity / State (in roles / dimensional points). 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).

Backend: RDF(S), OWL, Resource Services (Naming, Index Registry), ISO, Shapes, IDs matching. DIDs Backend.

Runtime: Objects, Events. Distributed Runtime Functional (Dataflow) reactive Resource objects / models (forms / flows).

Deployment: Spring / Vertx / others (Jersey / CDI: Resources Meta Model / Domain APIs). HATEOAS / HAL APIs (Encoding / Dialog Endpoint Protocol). Environment / Container: Messaging / Services. Models I/O.

Integration Connectors: Runtime Model embedded Resources. Sources as reactive stream objects. API.

Integration Clients: Runtime Model embedded Resources. Services as reactive stream objects. API.

Integrations endpoints (augment / extend) Service URIs APIs.

RDBMS.

Rules.

Data Virtualization.

OData.

Solid.

ERP / CRM ESB / EAI.

CMS / Feeds (Hypermedia augmented Resources) / Browser

Messaging (Connectors / Clients)

ML / Big Data.

StratML.

ISO 13250.

ISO 15926.

Dialog: Encoding / Protocol. Activation, Location, Context (Hypermedia type / context interfaces). Augmentations.

Encodings: XML / XSL / XPath / XLink / XForm / XPointer / XQuery (RDFS / OWL / ISO DM / RM Forms / Functors / DOM). Endpoints (streams / signatures): documents (messages) / resources (paths). JSON-LD.

Runtime. Bus. Signatures bindings (reactive), content type / domain context interactions data transforms (roles).

Design: Augmentation (Aggregation, Alignment, Activation Functors), Domain Functors. Encoding. Design Functors Dataflows.

CQRS. Event Sourcing. Functor Commands (HATEOAS browseable APIs).

Domain Driven Development:

DDD Connector / Client / Runtime (Hypermedia / Facade / Dialog Protocol: services / actions). Backend: Augmented Services / Persistence. Representations (resource: types / activation). Domain, Services, View Models.

StratML Client / Connector (Goods, Needs, Products, Goals, Purposes. Exchange ontologies).

ISO Topic Maps / ISO 15926 Facades.

# **Features**

## ***Domains schema***

Message based Augmentation Events Dataflow. Augmentation Mapping Dataflow allowing to embed merged dynamic state in Model entities (including Mappings Augmentations themselves). Reactive / event driven declarative application framework: Model, meta model, instances as streams sinks / sources (transformations).

## ***Ontology matching***

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).

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.

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

Ontology Matching: Encode: order, iteration, flows, units, relations, events, enums, etc.

Ontology matching: Dataflow: sort statements. Units. Equivalences. Distances / events (order). Services (Augmentation / Context Functors Meta Model mappings / transforms). OntResource; Merged URI(s) wrapper. OntResource hierarchy: layers statement contexts. Facets DOM, Actor / Role. Resource (OntResource Context Roles hierarchies Monad wrapper); Statement : Resource Role quad, Resource.

Ontology matching: Events declarative definition. State change of value in axis in measure of context. Dimensional Model.

Ontology matching (table, pk, col, val example). Helper upper models for models linking / alignment. Event sourcing (“offline” sync). Graph linking / alignment / sinchronization by entailments from event sourcing over inferred state. Reconciliation.

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. SortedSet hierarchies membership (octal) values.

Ontology Matching: Encode: order, iteration, flows, units, relations, events, enums, etc. Semiotic / Dimensional alignment. TBD.

Ontology Matching. Semiotic. Dimensional. Sets. Functional Reference Model. SortedSet hierarchies membership (octal) values.

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.

Ontology Matching:

Statements (encoding): Proposiciones / Razonamientos / Cuantificadores / Predicados / Clases / Relacion: validez / valor de verdad (en contextos, ejes, variables: funciones / enunciados / casos de sustitución / equivalencias). Pronombres.

Truth values: Predicate / reasoning. Sets encodings Predicate comparisons matches context templates / transforms mappings: inferences.

Relaciones de equivalencia, clases de equivalencia: reflexividad, transitividad, simetría. Identidad. Propiedades determinan alcanze / relato, dominio / codominio, universo y campo. Clases y relaciones (atributos / valores) determinan matcheos / relaciones entre clases / individuos y relaciones y tuplas / miembros. Formas proposicionales, categorícas (predicados), clases y relaciones (reglas, valores de verdad) para afirmar equivalencias (operar entre relaciones de propiedades).

Relaciones, cardinalidad: (1, n), (n, 1), (1, 1), (n, n). Función / inversa: inyectiva / biyectiva.

Relaciones de orden. Inclusión / jerarquías. Rendering: lattice (encoded statements / properties bitstring / vector). Equivalencias.

Predicate como occurrence de un Predicate: axis (attributes / values). Intensión / extensión (representante partición)

Ontology Matching: Dimensional metaclass / class / instance attributes relationships in axes / contexts / roles evaluations. Resolve equivalence of dimensions, units, measures, values via aggregation of value occurrences. Resolve value kinds by occurrences of Semiotic (grammar) model. Contexts (CSPOs) Attribute / Value "clustering". Sets specifications. Dimensional order relations.

Distributed Inference (dimensional / matching / predictions). Distributed consistency. Event sourcing (Dialog, CQRS Protocol). HATEOAS distributed / lazy reconciliation: hashing of source / destination (hierarchical contextual model quads hashing) state in interactions tokens. Prompts. Distributed Alignment (until reach of goal synchronization states). Distributed state / IDs.

Comparable: parent / child relations (SortedSet). Model sets / dimensional axes alignments.

Semiotic: syntax, semantics (grammar), pragmatics. Verbs: action, passion, state (for / due to action / passion). Models. Dimensional (axis / order) relations. Alignments. (Sorted) Set relations: hierarchical encoding. Verb roles (CSPO / Kinds sets).

Example aggregation: candy (type), red (color attribute), strawberry (flavor attribute).

(dado rango y alcance, universo: U de una relación: P, inferir dominio y codominio, campo: C). TBD.

From Object (O) extension / instances to Context (C) intension / class. Matching grammar shapes. Incrementally render type, role, occurrence context layers.

## ***Augmentations***

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.

Aggregation.

Alignment.

Activation.

Augmentation applied to Context: Aggregation.

Augmentation applied to Subject: Activation.

Augmentation applied to Predicate: Alignment.

Augmentation applied to Object: Role in context.

# **Deployment Use Case: Goals Application**

Products and Services Exchange Network (PASEN):

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 matching (Data, Schema, Behavior alignments):

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

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).

Use Case: Goals App

Goals App: purpose / goals / domain driven syndication of integrated business / social / cloud application features. User / Groups / Roles Purpose(s), Goal(s), Task(s) "intelligent" tracking oriented focus providing an abstraction and integration layer of players process flows / interactions and players process assets management and semantic orchestration.

Goals App: Semantically annotated gestures / interactions (contexts, purposes messages / interactions / resources / content). Subject context occurrence role attributes values (metaclass, class, instance, occurrences).

Goals App: API Facade for rendering aggregated data roles in contexts interactions topics / subjects assets (conceptual domain contexts axis / state views / activations: Forms / Flows). Example: domain declared Customer (actor / role), Product, Order, Purchase, Invoice, etc. topics / subjects assets rendered in contexts (Sales Report, Expenses Report, etc. embedded / linked dashboards). Wizards.

Goals App: Browse / search / activate: history / relations / referrer context / interaction / gestures roles traceability / (dialogs). Gestures / interactions (actor / asset, actor / actor). Wizards.

Goals App: Hypermedia contents APIs (embedded / embeddable resources: Semantic contextual Wiki / Apache Stanbol / CMS: hypermedia augmentation, knowledge / behavior maps). Integration: augmentation / sync backends / apps. Extension: services / APIs. Annotate / augment link content. DAV protocol (integration / extension facades).

Products And Services Exchange Network:

Contents / Features (Mision / Vision). Distributed consistent Knowledge Applications. Trust. Consistency. Event sourcing. Inferencing (of distributed state). Reconciliation.

Certify distributed Entity / Subject Identity / State (in roles / dimensional points). 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).

The idea of the project is to "augment" an ESB for EAI platform and to enable it allowing it to make "inferences" regarding which routes to use, "discovering" sources / destinations of an event message(s) which then it transforms / enriches according destination "semantics" and format(s).

This featuring the exposure of a generic facade which allows to see in an "homologated" view the applications or services and their data, schema and behavior (actions) that could be integrated into the tool.

Different integrated applications are enriched with this facade and with the events that, given the inferred routes and transformations, augments theirs data, schema and behaviors, invoking activities corresponding to each destiny semantics.

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.

**Notes:**

**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>

**Scrapbook:**

Message (Form / Flow) / Resource: Meta Model parent classes? Specification / Protocol. Signatures: Mappings Context Kinds. Possible Flows (Form), actual Augmentation (Flow).

Basic hypermedia browse / CRUD (HTTP verbs) bound Message functors compatible for all Resources (REST).

Resolve Message / dialog (CRUD) semantics via MetaGraph driven transforms (data / schema / behavior augmentation: dialogs).

Basic Message aggregation (Context Mapping): shift right mapped applied statement resources. Mapped resource context > instance (occurrence) of next layer message reified resource context.

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);

Events: Monads (IDs Contexts hierarchy instances), Functors (layers classes instances reifying model classes / domain instances from facets / levels). Augmentation: materialized Transform. Flow: Mapping possible Transforms. Browse / Apply (generic flows?). Dimensional.

Classes: Layers monads Contexts cllass hierarchy. Inputs resolves from wrapper containers to next layer occurrences (map forward), occurrences contexts collects matching result graph (reduce backwards). Map / Reduce: Graph key / value / properties encoding.

Contexts:

* ID<ID> : Reified matching URIs.
* Transform<ID> : Range
* Mapping<Transform>
* Template<Mapping> : Domain
* Augmentation<Template>
* Resource<Augmentation>
* Role<Resource> : CSPO Role
* Statement<Role> : CSPO Quad
* Model<Statement> : Set of Statements
* Layers:
* (ID, ID, ID, ID);
* (Transform, ID, ID, ID);
* (Mapping, Transform, ID, ID);
* (Template, Mapping, Transform, ID);
* (Augmentation, Template, Mapping, Transform);
* (Resource, Augmentation, Template, Mapping);
* (Role, Resource, Augmentation, Template);
* (Statement, Role, Resource, Augmentation);
* (Model, Statement, Role, Resource);
* Facets.

Functors: Model / Domain Augmentations (Forms / Flows). MDM. Provenance. Versions (time, dimensional / semiotic / functional axes: location, language, events, types, roles, behaviors, other rels).

Flows: Addressable interactions (Model signatures reactive bindings).

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

Messages: CRUD / Domain  Invocation semantics. Flow grammars / verbs. Dialog. Prompts. Inputs are aligned into Message and are applied to Mapping Template and rendered by Mapping Transform (class extension for Augmentation class intention).

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).

Component. Services. Protocols. Archetype Reactive Functional (Monads) Component APIs. Reactive: Connector / Client Endpoints: Consumer / Producer / Processor (Service / Model) inputs / outputs handlers (formats / protocols parsing / matching / alignment into IDs / Contexts. Reactive Augmentations: fire possible dataflows).

Environment: Models events abstraction (subscribe / augment / publish) Connector / Model / Client Augmentations IO.

Encoding: XML / XSL / Template Scripts (functional runat: peer dialogs / reactive callbacks). Mappings declarations / encodings (primitives, wildcards, variables, placeholders templates: actual / result of, possible).

Streams: URIs, Resource, Statement, CSPO Roles, Kinds. Dataflow: index / signatures dispatch, reactive.

Formalization: Functional / Object APIs. Reference / Data model. Sets, categories, models. SortedSet (hierarchical structures).

Kinds, Signatures. Contents. Contextual metadata. Lattices. Roles / Sets (bitstring cuads). Definitions (elements). SortedSet (hierarchical structures). Key / Value graph encoding. Map Reduce. Flows: Mapping declarations / assertions (possible flows).

Operations. Rules. Categories. Groups.

Semantic resolution: Query Resource(s) satisfying “criteria” (i.e.: Object(s) for predicate / Augmentation Mappings Forms / Flows) IDs by IDs resolution pattern: (Message applicable signatures : resolution result: Transform).

Query Resources by role in context.

Query Resources by attributes / values.

Query Resources by identity / type.

Streams. Subject Kind: Subjects stream. Object Kind: Objects stream. Predicate / Context Kind: Flow Signature. Stream (filter SO kinds).

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).

Messages: Dataflow Template matches signatures (Session level, enrichs Message with Model / Dialog prompts / contents alignments). Augmentation Functor applied over Message contents (Interaction level). Transform matching output signature emits (Session level, populated / prompts) output Message (parse Transform).

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.

Semiotic:

Subjects: attributes / values. Occurrences: contexts / roles.

(Context, Occurrence, Attribute, Value);

(Context, Sign, Concept, Object);

Metaclass, class, instance, occurrence.

Assert order / hierarchies / relations in dimensional axes. Containment (sets).

Messages: Service Context URIs: Signature for face recognition (image URI / resource : domain, detection / search results endpoint / placeholder : range). Others services: ML Classification, Clustering, Regression, Services Index, Naming, Registry. Presets "inferred" models and augmentation services (populated / online learning).

Augmented Semantic Content Types (img/xml;facesCoords).

Upper Ontologies. Load. Grammar level services (schema browse, possible flows query / browse). Message: wildcards, variables, placeholders.

Dimensional:

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

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

Assert knowledge: 1h -> 60min;

Assert: dom-lun-mar-mie-jue-vie-sab;

Assert: 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.

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. SortedSet hierarchies.

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

* (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?).

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

Ontology Matching: Encode: order, iteration, flows, units, relations, events, enums, etc.

Ontology matching: Dataflow: sort statements. Units. Equivalences. Distances / events (order). Services (Augmentation / Context Functors Meta Model mappings / transforms). OntResource; Merged URI(s) wrapper. OntResource hierarchy: layers statement contexts. Facets DOM, Actor / Role. Resource (OntResource Context Roles hierarchies Monad wrapper); Statement : Resource Role quad, Resource.

Ontology matching: Events declarative definition. State change of value in axis in measure of context. Dimensional Model.

Ontology matching (table, pk, col, val example). Helper upper models for models linking / alignment. Event sourcing (“offline” sync). Graph linking / alignment / sinchronization by entailments from event sourcing over inferred state. Reconciliation.

Messaging metamodel:

(Message, Resource, LHS, RHS);

(Interaction, Message, Resource, LHS);

(Role, Interaction, Message, Resource);

(Context, Role, Interaction, Message);

(Dataflow, Context, Role, Interaction);

Encoding:

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. SortedSet hierarchies (3 digit octal set membership values).

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

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

Quad encoding: Context relative IDs.

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.

Dataflow: Order, Forms, Flows (Signatures, 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.

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).

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.

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).

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.

Functors:

Functors: Meta Model declarations / Context classes / instance by hierarchies: declarative implementations of monadic functors (Levels: Augmentation / Domain Flows). 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.

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 / Mapping Transform).

Backend:

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

Messages: Events IO / Persistence: Saga Activation / Passivation populating Node local Quad store / persisting peers via DIDs (inference enabled distributed consistency) semantic (resolvable / discoverable) identifiers.

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.

Functor applied to context: Aggregation.

Functor applied to subject: Alignment.

Functor applied to predicate: Activation.

Functor applied to object: members traversal.

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

Grammars / Levels / Discovery (Model Forms / Flows Specifications / Protocols): Definitions: Quads, contexts, Kinds, Grammar / upper ontology as level / aggregation relationship. From data to dialog gestures. Augmentations aggregation, alignment, activation.

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

Quads Context / Object: class by intension / extension. Transform matches Context signature, filters by Object(s) extension. Resource(s) specification.

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

Deployment / Use Cases:

Purpose driven hypermedia activation:

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

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

Browser referring context (Work, Peter, Employee).

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

Core (upper / onto) Messages: Getters, setters, nav, etc. Domain Messages: raiseSal: setSal(sal \* increment); promotion: setPosition.

Event sourcing / tracking: married -> marriage occurred. Dataflow: Messages hierarchy. Aggregate contexts from coarse to fine grained  transforms (raiseSal -> setAttr, single - marryWith).

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 appplication (flatMap) over / into (S). Messages inferred / aligned, activated, aggregated according base message transofrms resources. Messages inferred from models / layers. TBD.

R : Model Context hierarchy.

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). Matches Augmentation Forms / Flows.

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

Service URIs (reactive clients / connectors):

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.

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. Encode behavior in statements / graph: Comparisons, order. Sort. Order (kinds hierarchy?) Pattern matching, iteration, jumps. Discovery: routes / signatures, next event in bus / graph. Dataflow.

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

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 (SortedSet).

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).

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.

Protocol:

Protocol: Input statements for querying augmented knowledge (Specification Forms / Flows). 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.

Dataflow embedding: Resources reifying global state. Specifications: Forms, Flows. Augmentation Dataflow: 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.

Source / Grammar / Pragma Levels.

Functional / Dimensional / Semantic Facets.

Reactive Entities: Resource, Model, Message, Kind.

Entities: ID (routes), State (ctx / rel pointers, occurrences). Streams, Dataflow (routes / bindings: addressing).

Transforms, Augmentation (functors / mappings).

Dataflow: Message / Model /

Augmentation / Model / Message.

Meta Model (Interaction Layer Augmentation Aggregated Model declarations: facets, levels, layer).

Meta Model Interaction Layer (Augmentation: aggregated Source, Grammar, Pragma Levels Mappings) Mappings render Data, Schema, Behavior Resources for Functional, Dimensional, Semantic Meta Model Facets layers.

Entity Kind aggregation (Statements) procedure example. Encode into Quads. Alignment and Activation Quads encoding.

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. 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.

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): Basic type inference. Applied over layers CSPO during Activation Augmentation.

Source / Session / Pragma levels. DCI. Data / Information / Knowledge. Syntax, Semantic, Pragmatic. Model state: Context (Resource : data), Kind (Grammar : schema), Dimension (behavior). Context Kind(s) signatures: Dataflow.

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). CQRS. Dialog (EAI pattern: Isis docs).

Augmentation: Context Resource Monads / 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).

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 statement CSPO roles.

Alignment: Infer layer missing / deducible attributes and values for CSPO Subjects.

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.

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. SortedSet hierarchies membership (octal) values.

Ontology Matching: Encode: order, iteration, flows, units, relations, events, enums, etc. Semiotic / Dimensional alignment. TBD.

Ontology Matching. Semiotic. Dimensional. Sets. Functional Reference Model. SortedSet hierarchies membership (octal) values.

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

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.

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.

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.

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

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).

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

Meta Model: Encode / reify Model(s) w./ Meta Resources and Model Context(s) hierarchies. Meta Model: Encode order, iteration, conditional flow. Dataflow.

Encoding: Kind hierarchies / Grammars

(CK, SK, PK, OK);

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.

Graph Execution Semantics: Dataflow by Context Kind domain (Subject Kind) / range (Object Kind) matching Forms / Flows. Ontology Matching. Upper ontologies. Primitives.

Encoding: Resource ID. Encoded Resource contents (signature / occurrence). Augmentation: Resource (SortedSet) 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). Compose IDs (hierarchical graph properties encoded string) from outer to inner resources (Context, Kind, Occurrence, Role, Resource). "Operable" IDs (ClassIDs / InstanceIDs: Meta Model reifications / occurrences).

Encode common upper Semiotic / Dimensional Model: Reference Model.

Encode Kind / Context hierarchies.

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.

Object occurrence of Predicate.

Sets. Quads. SortedSet.

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

Encoding: recursive resource quads encoding hierarchy, order, class, instance, attributes. Operate inferences over (upper) patterns (bitstring / lattice). Meta Model, Facets, Levels. Specifications: Signatures, Forms, Flows (encode events / transforms provenance).

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

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

Order: Common super type / kind / role / occurrences. SortedSet.

Augmentation: common super type inference / alignment: 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).

* 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.
* 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.
* 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.
* Encoding: Layers relations: Metaclass / Class / Instance. Subject / Occurrence / Role / Attributes / Values. Mappings declarations: Specifications, Forms, Flows.
* Subject (Resource) / Context (Statement) / Occurrence (CSPO instance) / Role (Kind) / Attribute / Value.
* Metaclass (Occurrence) / Class (Context) / Instance (Attributes / Values).
* 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):

**MetaModel:**

Meta Model:

Arrangement of layered typed CSPO quads statements in which each CSPO role / type plays the role of "aggregating" previous layer abstract knowledge into more concrete aggregated contexts (statements) instances until a "reference model" CSPO type arrangement is achieved, which is the root of all other layers hierarchy.

The purpose of this is to achieve some ontology matching capabilities over an upper abstraction set of layers and to enable a Functional Knowledge Base Interaction APIs integration / virtualization overlay for matching and consumption of distributed datasets via endpoints dataflows.

**Reference Model:**

ID : URL;

Occurrence: Context;

Context : ID (Context / ID : intension, Object / Occurrence, Sign / Kind / Metaclass / Attribute, Value / Role / Class : extension);

Root of Meta Model hierarchy.

**Notation:**

[LayerType] : [LayerSuperType] ([ContextType], [SubjectType], [PredicateType], [ObjectType]);

**Layers:**

(Relation, Statement, Kind, Resource);

Resource : Context (Resource, Resource, Resource, Resource);

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

Role / Class : Statement (Role: b, Statement, Resource / Attribute, Resource / Value);

Role / Class aggregating CSPO Resource (IDs) sharing Attributes for their Objects / Values.

Kind / Metaclass : Role (Kind: a, Role, Statement, Resource);

Kind: Aggregated similar Roles occurring as Resources (Object) in Statements (Predicate).

Relation / Entity : Kind (Relation, Kind, Role, Statement: c);

Rel type (Relationship) instance / bindings. An Entity (Relation: intension) and their Statements for its Kind / Role occurrences (occurrences: kinds / roles Relation plays in statements. Matching. Object: extension). Data (DCI)[1].

Mapping : Relation (Mapping, Relation, Kind, Role: b);

Rel players types / bindings scenarios. Information. Interaction (DCI)[1]. Mapping Role and Relation Kind: dataflow promoted types / order: relationships players domain / range. Entity alignment.

Relationship : Mapping (Relationship, Mapping, Relation, Kind: a);

Rel type declaration, player types. Knowledge. Context (DCI)[1].

Mapping and Relationship layer contexts are "calculated" (reifying) by Relation layer context kinds.

Semiotic Layer (ontology matching):

Value (Value, Value, Value, Value);

Sign : Value (Sign, Value, Value, Value);

Object : Sign (Object, Sign, Value, Value);

Context : Object (Context, Object, Sign, Value);

Reference Model:

Root of MetaModel hierarchy.

**Matching / Relations / Attributes:**

One of the intentions of having all this layered infraestructure is to be able to inspect "relations", being them "reified" into a Relationship construct, or being them single attributes and values for a subject enabling the possibility of "align" one into another for ontology matching purposes.

(a, b, c: Kind, Role, Statement): Reified Rel. to / from expanded Attributes / Values. Matching / roles (intension / extension).

Context DOM: parent / child; previous / next siblings; attribute / value (determined by CSPO roles). Class / instance DOM relation for parent / children layers instances.

**Ontologies:**

Context layers instances. Levels. Example: Dimensional ontology. Ontologies should be able to be built upon Reference Model layer CSPO types arrangements.

Dimension, Unit, Measure, Value.

Axis, Behavior, Flow (state change), etc.

Primitives: dimensional upper ontology. In / Out, Prev / Next, Pick / Drop, etc. Opposites. State change (current). Events, state flows. Marriage example.

**Message Dataflow:**

Relationship, Mapping, Relation streams / signatures. Messages: Context instances. Functional Knowledge Base Interaction APIs.

Aggregation: Browse / Transform.

Alignment: Inference.

Activation: Dataflow type (signatures). Message dispatch (domain / range ordered). Aggregation.

**Relations:**

Inference. Relation types: transitive, reflexive, simetric. Campo, alcance, dominio, rango, transform / function: infer / aggregate. Context functor / monad.

Inputs: (Context / Relation, PK, column, value);

Inputs: (Infer S Kind / Role, S, P, O);

Inputs: aggregate occurrences. Statement Context for each SPO as Occurrence with corresponding Attribute / Value (S: PO, O: SP, P: SO, etc.).

Inputs (infer rels): Part / Whole. SPO / OPS. Attribute / Value.

Inputs (infer rels): Containment. SPO / SPO. Parent / Children. Occurrences of Contexts of same Context layers (inherited Contexts). Example: (Mapping, Mapping) for (Relationship, Mapping). Super / Sub type Contexts instances relationships.

Inputs (infer rels): Order. SO Ps Domain / Range.

Input (infer rels): Event. Prev / Next state change. Type promotion.

**Reactive Functional Reified Metamodel:**

Transforms: Match Selectors. Hierarchy polymorphism. Contexts streams. Browse Metamodel. Context, Subject Selectors.

Transforms: Templates. Context instances (CSs) declaration / augmentation (POs) Selectors. Metamodel activation. Predicate / Object Selectors.

Encode Match / Template as Context. CS: Match, PO: Template Selectors. Apply Templates (role bindings / prompts) declaration / augmentation activation to matching selected CS streams.

Augmentation: Aggregation, Alignment, Activation Reified Match / Template dataflows. Reactive Model instances Match / Template dataflows.

Selectors:

Apply Role to Statement : Statement / Statement to Role : Role.

Apply Kind to Role : Role / Role to Kind : Kind.

Apply Relation to Kind : Kind / Kind to Relation : Relation.

Apply Mapping to Relation : Relation / Relation to Mapping : Mapping.

Apply Relationship to Mapping : Mapping / Mapping to Relationship : Relationship.

Apply Context to Relationship : Relationship / Relationship to Context : Context.

**Meta Model / Backend:**

Reference Model / Occurrences annotations matrix:

Attachment: show an example of a fully expanded set of last model layer (Relation) and its corresponding occurrences property graph annotations. Previous model layers may be annotated accordingly in respect to their ability to aggregate more abstract contexts (Resource, Context) properties.

Reference Model / Occurrences annotations matrix:

Lattice / FCA: Contexts / Resources. Objects / Attributes (Contexts instances axes). (X, Y): Z (for corresponding pair types / functional transforms).

Statements and annotations: FCA Lattice / FCA Contexts / Attributes. Objects / Attributes (Contexts instances axes). (X, Y): Z (for corresponding pair types / functional transforms). Typed calculus in FCA context development. Sets.

IDs / bitstring encodings. Algebraic / arithmetic activation flows / templates / transforms metadata encoded selectors. Vector Space Model quads polygon embeddings.

Interfaces:

ID : URL;

Context: ID;

Object : Context;

Sign : Object;

Value : Sign;

Labeled Property Graph annotations example. Augments Reference Model. Statement example, Statement context aggregates SPO annotations (Statement occurrences data in other Reference Model layers contexts):

Context (Context : Object, Object : Sign, Sign : Value, Value);

Resource : Value (Resource, Resource, Resource, Resource);

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

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

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

Relation : Kind (Statement, Role, Kind, Relation);

Relationship : Relation (Relationship, Role, Kind, Relation); \*

\*: Relationship: Aggregated Relation Statement Relation (Object) Roles / Kinds.

Hierarchy: render / process layers (Relation, i.e.) as Context, Resource, Statement, Role, etc. (upper layers) contexts (i.e.: reify Relation as Context, Resource, Statement, Role, Kind layer contexts). Context semiotic layer: aggregation, ontology matching.

Intension / Extension: S / O.

sub / super hiers, containment: P / O.

Relation reification: Relation statement object: relation instance. a: Role / b: Kind: relation ends. (a): Role reifying rel attrs / values. (b): Kind Resource reifying rel subject (rel players).

**Model:**

Message Events Bus.

Context Monad & type hierarchy (AST). CSPO parameterized types & aggregation (layers hierarchies specializations).

DOM:

Layer (Contexts): events producer / consumer (streams observer / observable).

DOM Parsing MetaModel: aggregate occurrences containment / hierarchies. Layers: subtype / supertype browse parent / children / siblings (order) and Attributes / Values relations.

Context::matchFilter(arg : Context): signatures / kinds stream predicate.

Context::applyMap(arg : Context): apply updates (CS Contexts / PO CRUD) matching filter predicate Context. Fires event bus messages.

Matching applies to meta-model signatures (internal aggregation, alignment and activation augmentations) and to domain / actual models signatures. Render new Attributes / Values and CSPO statements.

DDD: Declarative AST / Dataflow VM. Runtime. Encodings (layers / messages / activations). Event sourcing backends (Blockchain). Patterns (DCI).

RDFS / OWL / Graph Backend: Reified Metamodel. Labeled Property Graph. APIs.

Functional Meta Model Context Layers. Reactive streams. Match / Template Selectors. API.

Stream Resources Connectors (P2P Connector Bundles Context I/O). Dataflow. Backends / Augmentations / Endpoints model layers. APIs (reactive / events).

Layered abstraction levels streams options / menu semantics (REST). Forms / Flows. Browse, match / transforms high level APIs. HATEOAS CRUD / Flows.

Patterns. Input formats. Sample data.

**FCA / Concept Lattices**

Model context statements of an upper ontology and occurrence statements for each ontology concept occurrences using FCA (Formal Concept Analysis) and rules from a pseudo-grammar.

Context statements / Occurrence statements: Grammars. Concepts / objects hierarchies: CSPO statements concept types / kind rules / terminal instances. Productions: concept types / kind rules / terminal instances mappings / flows.

Aggregate kind rules (grammar) into context statements / Occurrence statements: polymorphically, Kind context applies to all Resource hierarchy (all lattices).

Contexts parsing: monadic parser combinators / monadic AST. Recognize context types from (surrounding) reified kind types / rules (link grammar). TBD.

Rules are of the form:

(TypeA, TypeB) > AggregatedKindResources;

For example, in Relation lattice:

(someRoleA, someKindB) > AggregatedRelationResources;

Dispatch: Model (lattices) observer / observable (streams / functional) of grammar (contexts case match) events.

Lattice (FCA Contexts) population and Augmentation:

Base layer: Context. Resource, Statement, Role, Kind, Relation layers.

Input layer: case match layer grammar type signature. Reactive dataflow dispatch inputs top-down / bottom-up contexts layers hierarchy (lower hierarchy layers polymorphically materializes upper hierarchy layers) for Augmentation.

Lower hierarchy layers contexts mapping / function transforms into next upper layer context (example: Roles to Kinds). TBD.

Contexts (TBD):

Data layer: Resource, Statement, Role, Kind, Relation contexts (aggregation).

Information layer (occurrences / interactions): Data layer contexts statements products as new contexts (aggregation).

Knowledge layer (Dimensional / DCI Contexts): Information layer contexts statements products as new contexts (aggregation).

Tensor like arrangements by FCA / grammars (TBD):

Aggregations: Data layer contexts containing Information layer contexts containing Knowledge layer contexts. Encoding: embeddings, object / attribute features bitstrings, VSM (Vector Space Model) quad polygon angles. Order encoding (lattice). Activation / Aggregation (CUD): preserve order (lattice), arrange input message as production of corresponding context shape. Retrieval / transforms: resolve production of corresponding context shape for input message.

Augmentation:

Activation: Layer receives matching context message.

Aggregation: Layer resolves productions of grammar rules for context message.

Alignment: Resolution of relevant knowledge and input message. Emits output message.

Notes:

Context occurrences statements (i.e.: Statement in Kind context). Occurrence contexts: (S, S); (P, P); (O, O);

FCA: Lattice. Ordered Sets. Intension / Extension.

To Do:

Model Order. Axes: dimensions, units, measures. Events (measures / relations / state boundaries in data / information / knowledge levels: price, price at moment in time, variation, tendence predictions, idem for distances, etc.).

Hierarchies: metaclass / role, class, instance, occurrence (parent, children, previous, next, attribute, value). Encoding. Comparisons. Functional traversal (streams).

Dimensional contexts: Contexts from Occurrences contexts statements. Dimensional contexts: Events (attributes). Order relations assertions by context occurrences hierarchy domain / range, set / superset attributes relations.

(Mapping, Kind, Role, Statement);

Event (Dimensional context attributes): (Mapping / unit / class, Mapping super / parent / dimension / metaclass, Kind unit / measure / occurrence, Role measure / value instance);

Model Application domains: upper ontology (Behavior, Flow, etc.) encoded in meta model and specialization levels for domains contexts. Declarative abstractions ontology for application design: discovery, alignment and matching for services renderings and integrations.

Hierarchy: render / process layers (Relation, i.e.) as Resource, Statement, Role, etc. (upper layers) contexts (i.e.: reify Relation as Resource, Statement, Role, Kind layer contexts).

Context (reified / instances) kinds (topics): Resource content types. Resource (monad): representation, HATEOAS, dialogs. Dataflows (order / domain / range kinds).

HATEOAS: Form / Flow. Operations / Dataflow Representation / State IO (CRUD) prototypes / templates. Prompts (values / operations). Dialog. Gestures. Context: navigation state (i.e.: pick operation value prompt shows value type Form). DDD DOM.

**Meta Model (DCI)**

DCI Meta Models:

Data (data): Relation hierarchy Model.

Context (schema / dataflow): Augmentation hierarchy Model. Model layers extends corresponding Data layers.

Interaction (behavior / services) Dialog hierarchy Model. Model layers extends corresponding Data layers.

Contexts and Interactions Models extending / reified as Data Model layers enabling matching, inference and augmentations (FCA / ML embeddings for example) for behavior and schema alignments.

**Meta Model (Data)**

**Reference Model:**

ID : URL;

Occurrence: Context;

Context : ID (Context / ID : intension, Object / Occurrence, Sign / Kind / Metaclass / Attribute, Value / Role / Class : extension);

Root of Meta Model hierarchy.

**Notation:**

[LayerType] : [LayerSuperType] ([ContextType], [SubjectType], [PredicateType], [ObjectType]);

**Layers:**

Resource : Context (Resource, Resource, Resource, Resource);

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

Role / Class : Statement (Role: b, Statement, Resource / Attribute, Resource / Value);

Role / Class aggregating CSPO Resource (IDs) sharing Attributes for their Objects / Values.

Kind / Metaclass : Role (Kind: a, Role, Statement, Resource);

Kind: Aggregated similar Roles occurring as Resources (Object) in Statements (Predicate).

Relation / Entity : Kind (Relation, Kind, Role, Statement: c);

Rel type (Relationship) instance / bindings. An Entity (Relation: intension) and their Statements for its Kind / Role occurrences (occurrences: kinds / roles Relation plays in statements. Matching. Object: extension). Data (DCI)[1].

Mapping : Relation (Mapping, Relation, Kind, Role: b);

Rel players types / bindings scenarios. Information. Interaction (DCI)[1]. Mapping Role and Relation Kind: dataflow promoted types / order: relationships players domain / range. Entity alignment.

Relationship : Mapping (Relationship, Mapping, Relation, Kind: a);

Rel type declaration, player types. Knowledge. Context (DCI)[1].

Mapping and Relationship layer contexts are "calculated" (reifying) by Relation layer context kinds.

Semiotic Layer (ontology matching):

Value (Value, Value, Value, Value);

Sign : Value (Sign, Value, Value, Value);

Object : Sign (Object, Sign, Value, Value);

Context : Object (Context, Object, Sign, Value);

Reference Model:

Root of MetaModel hierarchy.

**Dataflow Model (Context).**

Dataflow:

Iterations (types / kinds order declaration). Streams: Contexts / Occurrences Bus (signatures / discovery).

Conditionals (predicates / filters on types / kinds attributes / values) on Iterations.

Jumps (aggregation / stack sub-streams) on Conditionals. Apply Kinds on matching / referring Contexts (Employment, Person: Employee). Extract Kinds on matching / referring Contexts (Family, Father).

Order. Comparators: common upper hierarchies, Dataflow domain / range, SortedSet. Lattices (FCA contexts).

Augmentation: Aggregations, Alignments, Activations. Perform encoding dataflow.

Encoding: Augmentation, Template, Mapping, Transform.

OntResource: Ontology Matching (aligned URLs): semiotic context.

Dataflow Model:

Resource: (OntResource, OntResource, OntResource, OntResource);

Statement: (Transform, OntResource, OntResource, OntResource);

Role: (Mapping, Transform, OntResource, OntResource);

Kind: (Template, Mapping, Transform, OntResource);

Relation: (Augmentation, Template, Mapping, Transform);

Template: Reified Model and Model instances (hierarchies). Match inputs. Iterations.

Mapping: Dataflow reified operation flows bindings (subscriptions). Predicate / Object stream endpoints?. Conditionals.

Transform: Dataflow reified results. Jumps.

**IO / Services Meta Model (Interaction):**

Model for back ends synchronization and services exposures.

Context: (Model, OntResource, Resource, Resource); Key / value dictionary for source Model ontology matching.

Resource: (OntResource, OntResource, OntResource, OntResource); Full matched Resource descriptions: Type, ID, Attributes, Values.

Statement: (Assertion, OntResource, OntResource, OntResource); Transform (Jumps). Source Model(s) interaction interface Services URLs (IO).

Role: (Prompt, Assertion, OntResource, OntResource); Mapping (Conditionals). Predicates (LHS: Predicate, RHS: Object).

Kind: (Interaction, Prompt, Assertion, OntResource); Template (Iterations). Aggregate matching Interaction Assertion Prompts.

Relation: (Dialog, Interaction, Prompt, Assertion); Augmentation. State flows.

**FCA / Lattices**

For each layer context statement build tables which axes correspond to each context CSPO context types. Aggregate CSPO types / values in the form shown below.

FCA Context (tables): Context (Relation, Kind, Role, Statement, Context, etc.) matrices w./ corresponding CSPO x CSPO types axes. Types intersections determine cell type (as in example tables shown). Values intersections are instances of corresponding types. Example:

(RoleA x StatementB: KindC) : Kind RoleA plays in StatementB.

Thus, each layer context statements are used to build a matrix of CSPO x CSPO of its types and (scaled) values. In this manner (aggregating matrices / FCA contexts), SPO Resource occurrences in matrices axes SPOs / context layers statements SPOs are reified in statements for which the occurring SPO Resource is the statement Context and its SPOs are resolved according its SPO position in the original Context. This statements form the matrix rows and columns SPOs according some aggregation layout.

The purpose of this is to retrieve enough concepts (FCA) metadata to populate concepts / objects / attributes conforming a Lattice of related Resources and those relations values (as in the above example).

FCA Lattice (concepts / attributes / objects): (types / values) x (types / values). Encoding (IDs): ontology matching enabling type / instance calculations / traversal / transforms.

Relation matrix:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Relation | Kind | Role | Statement |
| Relation | Relation | Kind | Role | Statement |
| Kind | Kind | Relation | Statement | Role |
| Role | Role | Statement | Relation | Kind |
| Statement | Statement | Role | Kind | Relation |

Matrices for other layers (Kind, Role, Statement, Resource, Context) contexts follows the same principles.

Reifying one aggregated layer SPO layer (for example: Kind in the previous table) has original context matrix axes in the corresponding SPO layer (Subject in this case):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Kind | Relation | Statement | Role |
| Kind | Kind | Relation | Statement | Role |
| Relation | Relation | Kind | Role | Statement |
| Statement | Statement | Role | Kind | Relation |
| Role | Role | Statement | Relation | Kind |

The “generic” form of the table is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Context | Occurrence | Attribute | Value |
| Context | Context | Occurrence | Attribute | Value |
| Occurrence | Occurrence | Context | Value | Attribute |
| Attribute | Attribute | Value | Context | Occurrence |
| Value | Value | Attribute | Occurrence | Context |

Use cases:

Use FCA Lattice for sorting / ontology matching / augmentations / query / ontology browsing.

Aggregation: Complete contexts objects / concepts / attributes by FCA / inference.

Inference example: (Statement x Statement): Relations between both Statements.

Learning: ML embeddings for types / values / concepts.

TBD: (metaclass, class, occurrence, instance) relations / atttributes.

TBD: Set oriented intension (C) / extension (O) and relations between sets.

TBD: Discover IDs / encoding techniques enabling algorithmic translation of models operations.

Encoding: FCA Scaling. FCA Context objects and attributes are corresponding CSPO Contexts types scaling enclosed Context types instances. A potential encoding of axes objects and attributes (rows and columns) would be a bitstring of length 4 x n, being n the length of an instance identifier for each quad Context encoded in its corresponding bitstring quad space (4 is for CSPO quad types instances identifiers segments). Then, navigation should be allowed from a pair of object / attribute to another object / attribute: (type, object) x (type, object): (type, object).

If Context types / instances identifiers are sequential in form, a mapping (hashing) could be done in a bitstring of the length of scaled attributes (columns) having a 1 in the corresponding attribute for a given object. Attributes also may be rendered as a sequence of prime numbers being an object extension the product of its attribute primes.

Layout: The aggregated statements have as Contexts the occurring SPOs in a Context layer statement and its SPOs are the occurrence Context and the other SPOs in the occurring statement. For a Context in an aggregated statement occurring as (SPO) in the occurrence statement, occurrence statement Context is its (SPO) and its aggregated (SPO) is occurrence statement (SPO).

Layout: Having a Context layer, a matrix (FCA context) of the form (CSPO x CSPO) is built for aggregation of models. The aggregated statements (rows / columns) have as Contexts the CSPOs (occurrences) of an axis and and its SPOs are given from the types / values of the context layout. For an aggregated Context statements / matrix, the original context is located in the (SPO) axis from the (SPO) which it was taken from the original Context matrix.

Encoding: Layout rules. Context layers, use layout to aggregate occurrences matrix. Use aggregated occurrences matrix to retrieve original matrix.

Encoding: Layout. Layers. Aggregation: upper layers / lower layers. Encode Augmentations. Browse. Transforms. Reified Model. DCI.

Layer declarations: TBD (Context, Occurence, Attribute, Value) reified types / data.

Resource matching (reified / data): context / occurrence rules. Context population. Types / Functional / Data Models.

Layer downwards: Layer for which Subject is Context.

Layer upwards: Layer for which Context is Subject.

Facets: Concept hierarchies common attributes. Types: reified model objects instances: contexts / layers / aggregations. Values. Parsing (case match): resolve if an (scaled) type / value object corresponds to an (scaled) type / value attribute and which is its intetsection type / value (grammars / signatures / aggregation / dataflow). TBD.

Facets: cells Context type: from corresponding Kinds for reified CSPO roles of SPO axes intersections (type intension). Context values: CSPO Resources of Kind type extension.

CSPO / Kinds types / values (FCA scaled types / values objects / attributes intersections): Multiset (OrderedSet) encoding of CSPO Statements and Kinds as bitstring quad (CSPO segments) for corresponding sets / elements: C, S, P, O Resources arranged in a three SPO sets distribution (diagram) with SP intersection for Object Kinds, PO intersection for Subject Kinds and OS intersection for Predicate Kinds (Kinds being referencing SPOs in their corresponding parts and aggregating Kind values in their Kind type part).

Kinds layouts:

(S, P): OK; (P, O): SK, (O, S): PK; (Idem, Idem): Contexts, i.e.: Relations of equivalent Context signatures.

CSPO Sets Context (data layer, context / interaction hierarchies reification) population: parsing over RDF quad statements IO. Create / aggregate quad bitstring Resource / Kind identifiers. Functional query / browse / transform. Populate FCA contexts / lattices and calculate cells / dataflow (sets streams).

Populate Contexts (Sets, FCA, layers): Resources, Statements: inputs.

Role (CSPO Context hierarchies reified roles), Statement, Resource, Attribute. From sets aggregation.

Kind, Role, Statement, Resource. From sets Kinds aggregation.

Relation, Kind, Role, Statement. From sets reified Kinds / Context / Statements aggregation (Kinds domain / range).

**Index3:**

**Contents: Mision / Vision**

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.

**Solution Approach**

Objectives:

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

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

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

Declarative Application Design.

Features / Approach:

Data / Schema / Behavior Abstraction:

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

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

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

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

Ontology matching (Data, Schema, Behavior alignments):

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

Augmentation:

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

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

Alignment: Infer layer missing / deducible attributes and values.

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

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

Protocol (deployment):

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

Applications:

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

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

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

**Use Cases**

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

Semantic components:

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

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

ToDo.

**RDF 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 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.

**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.

ToDo.

**Deployment**

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

**Index2:**

**Contents / Features (Mision / Vision). Distributed consistent Knowledge Applications.**

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).

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).

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.

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).

**Index:**

**1: Mision / Vision**

**1.1: Objectives**

**1.2: Description**

**2: Use Cases**

**2.1:  Problems**

**2.3: Solution**

**3. Approach**

**3.1: Services / Features**

**3.1.1: Reactive / Event Driven**

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

**3.1.2: Augmentation Dataflow**

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.

**3.1.3: 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.

**4: RDF Introduction**

**5: RDF Quads / Objects Mapping**

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).

**6: Models**

Source / Grammar / Pragma Levels.

Functional / Dimensional / Semantic Facets.

Reactive Entities: Resource, Model, Message, Kind.

Entities: ID (routes), State (ctx / rel pointers, occurrences). Streams, Dataflow (routes / bindings: addressing).

Transforms, Augmentation (functors / mappings). Dataflow: Message / Model / Augmentation / Model / Message.

Meta Model (Interaction Layer Augmentation Aggregated Model declarations: facets, levels, layer).

Meta Model Interaction Layer (Augmentation: aggregated Source, Grammar, Pragma Levels Mappings) Mappings render Data, Schema, Behavior Resources for Functional, Dimensional, Semantic Meta Model Facets layers.

Model Layers:

Data Layer (aggregate Message in Interaction Mapping Augmentation context).

Schema Layer (aggregate Data).

Behavior Layer (aggregate Schema).

Model Levels:

Source Level (Functional Facet).

Grammar Level (Dimensional Facet).

Pragma Level (Semiotic Facet).

Model Facets:

Functional Facet.

Semiotic Facet.

Dimensional Facet.

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.

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).

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).

**6.1: 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.

**6.1.1: Interaction Layer**

(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).

(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.

**6.1.1: Data Layer**

**6.1.2: Schema Layer**

**6.1.3: Behavior Layer**

**6.2: 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.

**6.2.1: Functional 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);

**6.2.2: Semiotic / Semantic 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);

**6.2.3: Dimensional 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);

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.

**6.3: 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.

**6.3.1: Model Source Level (Backend)**

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

**6.3.2: Model Session Level**

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

**6.3.3: 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).

**6.4: Model Monad**

**6.5: Functional APIs**

**6.6: Streams (Contexts, Kinds, filters)**

**6.7: Transforms (Meta Model).**

**7: 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.

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:

Object: class (extension);

Context: super class (intention);

(ID, ID, ID, ID);

(Transform : Message, ID, ID, ID);

(Mapping : Verb query / assert, Transform : Message, ID, ID);

(Template : Message, Mapping, Transform : Message, ID);

(Augmentation, Template : Message, Mapping, Transform : Message);

(Model, Augmentation, Template, Mapping);

(Resource, Model, Augmentation, Template);

(Role, Resource, Model, Augmentation);

(Statement, Role, Resource, Model);

(Kind, Statement, Role, Resource);

(Class, Kind, Statement, Role);

(Context, Class, Kind, Statement);

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

Meta Resources layers contexts class hierarchy:

Context : Class : Kind : Statement : Role : Resource : Model : Augmentation : Template : Mapping : Transform : ID;

Meta Resources layers Reification: Context : Class : Kind : Statement : Role : Resource : Model : Augmentation : Template : Mapping : Transform as ID Context Statement (ID statements for each context layer). Model Levels (Facets / Levels). Aggregate reifications.

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.

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.

Functional API: Message IO. Mappings.

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.

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).

**7.1: 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 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).

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.

**8: Resource APIs**

**8.1: Resource Monad**

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

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.

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

**8.2: Functional APIs**

**8.3: Streams (Occurrences, etc.)**

**8.4: Transforms (Augmentation)**

**9: 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.

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

**9.1: Message Monad**

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).

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

(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: 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.

**9.2: Functional APIs**

**9.3: Streams (Signatures)**

**9.4: Transforms (Mapping)**

**9.5: Persistence**

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.

**10: IDs: Addressing / 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: ID

B: Transform

C: Mapping

D: Template

E: Augmentation

F: Model

G: OntResource

H: CSPO Role

I: Statement

J: Kind

K: Class

L: Context

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

Meta Resources layers contexts class hierarchy:

Context : Class : Kind : Statement : Role : Resource : Model : Augmentation : Template : Mapping : Transform : ID;

Meta Resources layers Reification: Context : Class : Kind : Statement : Role : Resource : Model : Augmentation : Template : Mapping : Transform as ID Context Statement (ID statements for each context layer). Model Levels (Facets / Levels). Aggregate reifications.

Meta Model:

A: (ID, ID, ID, ID);

B: (Transform, ID, ID, ID);

C: (Mapping, Transform, ID, ID);

D: (Template, Mapping, Transform, ID);

E: (Augmentation, Template, Mapping, Transform);

F: (Model, Augmentation, Template, Mapping);

G: (Resource, Model, Augmentation, Template);

H: (Role, Resource, Model, Augmentation);

I: (Statement, Role, Resource, Model);

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

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

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

ID: (L (K (J (I (H (G (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.

ID: Augmentation occurrences metadata in Statement contexts;

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.

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

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))));

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.

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 - 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).

**11: Dataflow**

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.

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.

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).

**11.1: Dataflow (Streams)**

Signatures: Routes (bindings).

Augmentations: Transforms.

Message

Model

Resource

Model

Message

**11.2: Model Interaction Layer matches Messages**

**11.3: Interaction Layer (Augmentation Dataflow)**

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

**11.5: Signatures. Addressing / Discovery. Bindings.**

**12: Augmentation (Transforms)**

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

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).

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).

**12.1: Augmentation Functor(Message, Message) : Events**

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.

**12.2: Augmentation Functional APIs**

**12.3: Streams (Dataflow Model Events declaration)**

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

**13: 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.

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.

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.

**14: Deployment**

**14.1: Bus**

**14.2: Service**

**14.3: Connector**

**15: Services**

Services. Connectors. URIs APIs. Endpoints (Events Mapping) messaging interface.

**15.1: Registry**

**15.2: Index**

**15.3: Naming**

**16: Default Augmentations**

**16.1: Aggregation**

**16.2: Alignment**

**16.3: Activation**

**17: Implementation**

Persistence

Messaging

Bus

Service

Connector

**18: Client Platforms**

Connector Activation.

**19. To Do**

Augmentation: Aggregation, Alignment, Activation.

Meta Model. Facets. Encoding. Augmentation: Functional, Semiotic, Dimensional inference / learning.

Ontology Matching: cross Facets Augmentation over Reference / Meta Model.

Model Facets: Predictions. Meta Model Facets possible Augmentation(s).

Map Reduce: Augmentation. Mapping (map) Transform. Streams (routes bindings). Template key, Transform value.

Encoding: Occurence(s) Context awareness. Kinds / Roles. Meta Model / Augmentation / IDs.

Model APIs. Monadic / Functional:

Dataflow: Mappings. Augmentations. Streams: Monadic chaining from chained sources / events. Signatures (Context Kind) domain / range route bindings.

Monad: Quad. Reified CSPO Statement: (Context, Occurrence, Attribute, Value); Parent Augmentation Monad.

Layers (Aggregation fmap):

(Augmentation, Model) : Resource.

(Model, Resource) : Role.

(Resource, Role) : Statement.

(Role, Statement) : Kind.

(Statement, Kind) : Class.

(Kind, Class) : Flow.

(Class, Flow) : Behavior.

Augmentation (Alignments fmap):

(Model, Augmentation) : Template.

(Augmentation, Template) : Mapping.

(Template, Mapping) : Transform.

(Resource, Model) : Augmentation.

(Role, Resource) : Model.

(Statement, Role) : Resource.

(Kind, Statement) : Role.

(Class, Kind) : Statement.

(Context, Class) : Kind.

Types: Roles (CSPO) Kinds (Activation fmap):

(Statement, Role) : Kind. TBD.

MDM. Provenance. Versioning. Dimensional context values / queries. Model Facets APIs: Functional, Semiotic, Dimensional Dataflow contexts / order / roles HATEOAS APIs workflows. Protocol: Dialog. Browse / analyze / transform "activations" (REST / JAF) according Facets.

Deployment:

Jersey / CDI. NodeJS. Others.

Protocol (Events / Encoding : Augmentation / Model). APIs / Services interface. Node: Signature routes bindings. Bus. Message I/O: Model objects parsing / rendering (Augmentation / Backend).

Models. Functional object models APIs. Messages (Augmentations, Models. Dialog: Protocol).

APIs: Protocol based declarative Model interaction interfaces. Endpoints: CRUD, ETL, Connectors, Clients, OGM / DOM. Domains activation modelling.

Services: Protocol based distributed declarative interaction interfaces. Protocol integration layer.

Naming: Resolve distributed names / matching identifiers contexts. Aggregation.

Index: Search distributed identifiers by attributes. Augmentation. Alignment.

Registry: Bound identifier context attributes / lookup identifiers roles. Activation.

Backend: Model Indices (Context layers, Statement Roles, IDs): combinations / permutations / rotations. Filter (Map Reduce queries: match patterns). Augmentations. Quads: rotations, index layers occurrences. Encoding IDs: tuple(s), permutations. Permutations / rotations: prefix permutations of indexed tuple.

Backend: Dataflow index. Index Augmentation(s) / Mapping domain / range signatures patterns ordered bindings (streams / routes). Mapping Rules: Grammar, Combination transforms. Order. Repetition.

Backend Quads (layers): Augmentation (Context) input given to Template (Subject) : Mappings.

Backend Quads (layers): Model (Context) input given to Augmentation (Subject) : Templates.

Backend Quads (layers): Resource (Context) input given to Model (Subject) : Augmentations (Resource is result of Augmentation instance in Model).

* ID<ID>
* Context<ID>
* Message<Context>
* Template<Message> : Augmentation Mapping range
* Mapping<Template> : Functional Transform. Resource materialized results (query / assert: Dialogs)
* Transform<Mapping> : Augmentation Mapping domain
* Augmentation<Transform> : Resource is Augmentation Mapping Transform result / instance
* Resource<Augmentation> : Augmentation Transform Mapping materialized result
* Role<Resource> : CSPO Role
* Statement<Role> : CSPO Quad
* Model<Statement> : Set of Statements. Facets (Mappings)
* (ID, ID, ID, ID);
* (Context, ID, ID, ID);
* (Message, Context, ID, ID);
* (Template, Message, Context, ID);
* (Mapping, Template, Message, Context);
* (Transform, Mapping, Template, Message);
* (Augmentation, Transform, Mapping, Template);
* (Resource, Augmentation, Transform, Mapping);
* (Role, Resource, Augmentation, Transform);
* (Statement, Role, Resource, Augmentation);
* (Model, Statement, Role, Resource);
* Tryton
* GNU Health
* Apache Metamodel
* JBoss Teiid
* JBoss Drools / JBPM
* OData
* R2RQ
* SPARQL
* ISO 15926
* DCI / MVC: JDBC / OGM / ORM / JCA / Activation JAF / Process Flows (state)
* Declarative hypermedia: REST / HAL / HATEOAS
* Declarative hypermedia: SOAP / WSDL
* CMS / Wiki (API / Protocol / DAV). Docs. Forms (Docs Flows)