**Encoding:**

IDs:

URIs.

IDs:

A: OntResource.

B: CSPO Role.

C: Statement : OntResource Occurrence.

D: Kind CSPO Instances.

E: Class : Kind CSPO Classes.

F: ContextStatement : Context Role.

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. 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.

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

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

URI(s);

OntResource; Merged URI(s) wrapper.

Resource (OntResource CSPO / Context Roles hierarchies Monad wrapper);

Statement : Resource quad, Resource.

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.

Encoding: Template Message augmentation (inputs).

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

Encoding: Transform Message augmentation (outputs).

Interaction Model:

(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.

Resource (OntResource Context Roles 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.

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.

**To Do:**

* 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).

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