# Goals

Achieve systems integration via Connectors: Smart ESB (domain / range signatures metadata), provided Data, Schema and Behavior alignments via CSPO Resources, Kinds and Contexts matching. Encodings.

Consume diverse Connector backends and align to a common ontology. Extract Data, Schema and Behavior for each Connector alignments. Augmentations.

Expose unified Functional REST HATEOAS Facade of Encoded and Augmented backends. Connectors handle event driven backends synchronization. Services Functional Dynamic Object Model.

Connector API: Data, Schema, Behavior (CSPO Resources, Kinds, Contexts) inter Sail Events Bus. Connectors: Smart ESB (domain / range signatures metadata).

What is needed to do and have now is a kind of "semantic hashing" that allows to obtain the relationships between identifiers and their contexts embedded in the URNs: <https://www.w3.org/TR/did-core/> (Semantic Hashing).

Then, the use case is to do Domain Driven Development (isis.apache.org is a good example) but in base (declaratively based) upon, for example, a database dump or the description of a serie of services or application to integrate APIs.

From the metadata embedded, for example, in the dump's data and schema or APIs it could be "discovered" what is "about" the application to integrate in its domain (forms, workflows, etc.) and expose it as services to consume in a synchronized manner with the original application.

Upon this, instances of different "learnt" domains are able to be "merged" and to contemplate integration of schema and functional in a single platform and uniform API in a single facade / client with a REST HATEOAS pattern (MVC / DCI).

Features / Problems to solve:

Type inference, ontology (data, schema, behavior) matching, aggregation, order. Not finding clearly defined how to aggregate or sort quads / triples.

N-ary Relations: Implement how are descripted in: <https://www.w3.org/TR/swbp-n-aryRelations/> (ISO15926) in a DCI pattern.

Then, actions available or performed in the various Connector integrated domains Facade are exposed / reflected / synchronized in such a way that actions in one domain (i.e.: ERP) reflects / trigger actions in other domains (i.e.: HR, CRM, SCM, BI: Big Data from Sails inferences.

# 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.

# Encoding

**Metamodel:**

Object Model. To do

Upper Ontology Roles Aligned Object Models. Templates.

Roles: Metaclass, Class, Instance, Resource: Instance Occurrence, Kind, Context.

Serialization (Alignement Quads):

(Metaclass, Class, Context, Instance);

(Instance, Context, Kind, Resource);

(Resource, Instance, Context, Kind);

Layer APIs: Meta Model Bus. Events sourcing. Connectors. Templates / Dataflow sync.

**Encoding:**

Object Model. To do

Serialization (Aligned Quads):

(URN, Context, Kind, Resource);

Layer APIs: Addressing, Matching, Persistence.

**Augmentation:**

Object Model. To do

Serialization (Aligned Quads):

(Metaclass, Class, Attribute, Value);

(Class, Object, Attribute, Value);

(Object, Class, Attribute, Value);

Layer APIs: Aggregation, Alignment, Activation:

* Data (Data): key / value. Column: (price: 100). Data Aggregation Augmentation.
* Information (Schema): Record (keys / values relation): (price: 100, brand: ACME). Schema Alignment Augmentation.
* Knowledge (Behavior): Records (columns values relationship). Price variation behavior example: ((price: 100, brand: ACME, date: today, priceVariation: 0), (price: 110, brand: ACME, date: yesterday, priceVariation: 10)). Activation Augmentation: materialize relationships / facts.

**DDD REST HATEOAS DOM:**

Object Model. To do

Serialization (Aligned Quads):

(Metaclass, Class, Attribute, Value);

(Class, Object, Attribute, Value);

(Object, Class, Attribute, Value);

Layer APIs: Naming, Registry, Index.

Messages: Dynamic Object Model Functional Monads bound (kinds signatures subscriptions) Functions. Contexts (Data, Schema, Behavior) browse traversal / transform. Resource aggregates Messages.

(Resource, Transform, Mapping, Statement);

Connector Bus API. Messages.

Layers Dataflow Layout. Messages.

Templates: Activation. Messages.

**API Transactions Log**

Message: Transform Log. To do.

ResourceURNs: Uniform identifiers across occurrences. DID URN. Endpoint. ResourceURN Statements: uniform functional metadata (contextual type / name, relations / aggregated occurrences). IDs Encodings.

Templates / Monads / Sets Interfaces. Graph layout. Traversal (Quads Monads). Set Membership Function: Interfaces CSPOs Types Matching Signatures.

Interface types differentiate in their CSPOs return value types (CSPOs type signatures: sets membership function).

Input / Canonical: Match Interfaces / Signatures: (Context, Occurrence, Attribute, Value);

Attribute / Value Roles in matching interface context. Order / hierarchy encoding: functions (sorted wrapped functional collections: wrappers set comparators / aggregation in axis).

Interface Quads Matching determine Sets (intersections) membership.

Hashing: IResourceURN, IOccurrence, IKind, IResource. Nested recursive URNs aggregations. Aggregations / Order / Mappings / Traversal APIs.

ResourceURN Occurrence, Kind, Resource Bindings Augmentation Service APIs:

* Inputs: Aggregate SPO into CSPO: Aggregates Contexts Type / Table / Class Kinds. Aggregate PK Cols, Cols : Occurrence, Val : Resources.
* Inputs (Rel / Graph): (Type / Table / Class, PK : Resource, Col : Occurrence, Val : Resource).
* Inputs (Rel / Graph) FKs: Val : Resource equivalent PKs.
* Augmentations / Transforms:
* Data matching. Resource equivalence: identity / comparisons / order transforms.
* Schema matching. Predicates equivalence. Domain / Range types. Order: data flow contexts.
* Behavior matching. Domain / Range values applied functional predicates identity. Order: data flow interactions.
* Transforms: DOM Resource Roles traversal functions, i.e.: Resource Roles functional getters.
* Resource::context::metaclass::occurence::role

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.

## Encoding Layers

* Resource Hierarchy Categories: Resources, Kinds, Statements, Mappings, Transforms.
* Resources:
* C: Contexts (Transforms / Class), S (Occurring Resource), P (Mapping / Transform), O (Occurrence Resource) Resources.
* Kinds:
* Kinds: SubjectKind, PredicateKind, ObjectKind. Peter valueOf Employee Kind Category Instance.
* Kinds (Wrapped Types / Class). Encoded in Statement Context (Transform / Class). Resolvable Resource Types: Subject SubjectKind (Subject POs) Kind Statement Context.
* Augment Resources with Kinds in Context. Core Model Transforms Mappings Instances / Roles: Kinds and Singleton (Resource) Class.
* Statements:
* Statements : Performed / Matching Contexts Transforms.
* Statements Resources: C: Transform / Class, S (Occurring Resource), P (Mapping / Transform) /  O (Occurrence Resource). S/O. Values: Resources / Reified
* Values. Resources Model / Domains Objects Hierarchy. Activation Augmented Types. Alignment Domain APIs (Measures, Dimensions, etc.).
* Mappings:
* Statement Predicates.
* Context / Statement Transforms / Mappings:
* Dual Occurrence of Transform (Mapping) / Occurring of Mapping (Transform).
* Transforms:
* Context / Transform / Class: (Context, SK, PK, OK). Performable Mappings Templates from learnt input / inferred Statements.
* Reification: Augmented Model.
* Encoding: Representations: Instances / Literals Encoding. URNs. Resolution: sameAs Mappings / Parsing. Occurrence / Occurring domainOf / rangeOf Type Inference.
* Kinds: Dimensional data, type, behavior / roles (dimensions) inference (declarative definitions / templates) by aggregation of occurrences (metaclass), attributes (class), values (instances). Sets / quads encodings (semiotical, context / occurrence / attribute / value, recursive statement / kind / SPOs).

### Addressing Sail

* Triples / Quads Tensor encoding shapes: triangles / squares (points in dimensional space, recursive from parent context shapes). Operations / activation functions: based on points / sides lengths / angles operations. Vertex position: seed / bias. Shapes compose graphs (arcs: sides / nodes: vertexs: points given each context / parent Tensor). SPO Vertexs. Context vertex (square inside triangle). Nodes containing sub-graphs (as contexts). Similarity: euclidean distance. Operations: order in contexts.
* Inputs: Aggregate SPO into CSPO: Aggregates Contexts Type / Table / Class Kinds. Aggregate PK Cols, Cols : Occurrence, Val : Resources.
* Inputs (Rel / Graph): (Type / Table / Class, PK : Resource, Col : Occurrence, Val : Resource).
* Inputs (Rel / Graph) FKs: Val : Resource equivalent PKs.
* Augmentations / Transforms:
* Data matching. Resource equivalence: identity / comparisons / order transforms.
* Schema matching. Predicates equivalence. Domain / Range types. Order: data flow contexts.
* Behavior matching. Domain / Range values applied functional predicates identity. Order: data flow interactions.
* Transforms: DOM Resource Roles traversal functions, i.e.: Resource Roles functional getters.
* Resource::context::metaclass::occurence::role
* Aggregation Statements Types:
* (Kinds / Types, Object / Resource / Predicate / Arc, Object / Resource);
* (Subject / Context, Kinds / Types, Predicate / Arc, Object / Resource);
* Reified Dimensions / Relationships.
* Roles.
* Reification. Infer Kinds / Types vía repeated matching of Statements Attributes occurrences. Quads Encoded. Attribute relation: Mapping from and edge of whim an Arc / Predicate is source.
* Encoding: Reification of Statement types to canonical form:
* (Context, Subject, Attribute, Value);
* Aggregate / de Aggregate mappings expansions.
* Operations: Monads, Transforms, Dataflow. Templates / Mappings for processing representations. To Do.
* Hashing: Common Metamodel. From source Connector URNs.
* Facade URNs: REST message driven bus (HATEOAS) Connector endpoints. Functional Augmentation aggregated DOM nodes transforms / traversal dataflow.
* Hashing: DIDs, CANs, CAMs, DLTs (Events). Content negotiation: signatures, headers (referrer: state browsing, E-Tag: state hash, facets: URN Dimensions).
* Relationship: PK of SKs, OKs, roles (metaclass of S/K occurrences). N-ary relations. Hashing: masks: traversal / functions / relations / operations (ternary CAM, 2 bits XOR: lattice encodings). DCI Contexts, FCA lattices dataflow.
* Connectors message / aggregated streams events dataflow layout:
* Aggregation: Statements for each Context Occurrence Attribute / Value. Mapping: for each matching Attribute / Value apply Transform, render Statement. Context: Transform / Class. Occurrence: Subject. Normal Form.
* Alignment: Aggregate Resources Context Occurrences, Attributes, Values for Resource, Kinds, Statements, Mapings, Transforms Resources from Statement, Mapping, Transforms occurrences / occurring. Positional Roles: Functional APIs: Resources Roles Reification. Wrapper Types. Aggregation: Matching.
* Aggregate Resources, Kinds, Aggregate Statements, Aggregate Mappings, Aggregate Transforms, Aggregate Context, Aggregate Subject, Aggregate Predicate, Aggregate Object (functional contexts). Transform Mapping Data Flow.
* ResourceURNs DIDs:
* URN: DIDs. Endpoint APIs: Statements types / sets (Resource, Kind, Statement, Mapping, Transform OntResources hierarchy) content types / classes: Functional APIs. OntResource (DOM DTOs) quads representations references other DIDs, handle resolution, interactions, etc. via other DIDs endpoints and Resource Monad API.
* Method: did:ont:[ID]
* ID : OntClassName (Sets) ":" [HashedQuad];
* HashedQuad : [HashedURN] ":" [HashedURN] ":" [HashedURN] ":" [HashedURN];
* HashedURN : "[" HashedQuad "]" | HashedCSPOString;
* HashedCSPOString : Context ":" Subject ":" Predicate ":" Object;
* URN::ontResource (traversal parsed representation).
* OntResource::URN.
* Encoding: methods
* Hashing: four segments identifiers. Sets, binary octal digit order operable hashing (4 bit per segment). Aggregation: Statements graph layout. Occurrences. S-Expressions, MonParsec, CoSQL, map-reduce.
* URN: Encoded quad. Hashing: traversal, discovery, resolution. Merkle tree (DLT / Events). Encode typing / naming in context, about DID State Statements (hashing metadata):
* Occurrence ResourceURN : (ResourceURN, Occurrence, Kind, Resource);
* Kinds: Aggregate Attributes.
* State: Aggregate Kinds Resources Attributes / Values.
* Hierarchy: Kinds Attributes set (super) subset (sub) Kinds relationship.
* Order. Aggregation: Kinds / States lattice / tree. Populate / encode ResourceURNs order in contexts.
* CoSQL. Monadic Parser Combinators. Content addressable RDF (S-Expressions). DIDs URNs. Zippers. RDF\*.
* Augment Resources with Kinds in Context. Core Model Transforms Mappings Instances / Roles: Kinds and Singleton (Resource) Class.
* Process: Inputs. Core Model (Occurrences / Occurring) Statements From Connectors / IO CSPO:
* Raw CSPO Inputs / Outputs:
* (Class, Instance, Attribute, Value);
* Model Statements I/O.
* Core Model Templates:
* Resources / Predicates Type Inference. Wrapped Types. Transforms / Mappings Types Aggregation. Align. Matching.
* Assert: (Class / Transform, Resource, Attribute / Mapping, Resource);
* Query: (Class / Mapping, Resource, Attribute / Transform, Resource) : CSPOs;
* Layers: Aggregate Inputs into Resources Matrix (Occurences / Occurrings Matrix).
* Perform Augmentations: match / apply Augmentations on each Layer.
* Materialize Augmentations (Occurrences / Occurring Statements). Mappings Transforms Resources Occurrences / Occurring Statements (Models) Results.
* Reactive Data Flow: Process new Models Inputs. Sources: Connectors / Peers, Results Feed Back. Event Sourcing (Models Subscriptions).
* Assert / Query Expansion: State Flows into Contextual Mappings / Transforms.
* Product / good / need: exchange token balance function (request / response exchange flows). Game theory.
* Classification / identification: data. Classification (sets).
* Attributes / links classes / instances: schema. Clustering.
* Roles in contexts classes / possible states / flows / instances: actual performances: behavior (activation, injection resolution). Regression (discrete encoded values for metaclass / class / instance in contexts).
* Encoding: action / pasion / state ctx quads of dereferenceable (applicable resources) IDs of IDs. Relations, predicates, functions, operations (injection resolution / signatures dataflow activation).
* Annotations / contexts schema. Reference 'events': compare alignments (class, attrs, roles) in rel to a temporal (octal) comparison.
* Cyclic ternary numbering: digits order relation a, b, c: a < b < c < a. Number (positional digits) weights calculation (preserve digits order rel in numbers).
* Number sets: elements, functions, relations, operations. Alignments (class, attrs, roles): encode as discrete (embeddings operable via grammars) values.
* Encoding: dimensional 'lines' for each order position. Lattice. Resolve encoding of dimensional data (instance), schema (class), behavior (metaclass).
* Unique IDs. Distributed / addressable exchanges: source, payload, destination 'validations'. State, versioning, alignments validations (possible state, local / remote 'hash' agreement).
* Ternary order semiotic positions: object < sign < concept < context < object.
* Ternary 'primitives’: self < this < that < the < self
* Quad (embeddings oprable): (Context, Concept, Sign, Object);
* Valid IDs: composite (positional) numbers such that order rel holds for ID in respect to its context.
* Alignments: production / validation of valid IDs (classification, attrs retrieval, ctx role regression).

### Matching Sail

* Encoding / Matching:
* Functional Primitives: (Matching). Graph Shapes Model. Layers. Example: reify / render / match "uncle" relation / "marriage" situation from graph statements.
* Aggregation Statements Types:,
* (Kinds / Types, Object / Resource, Predicate / Arc, Object / Resource);
* (Object / Resource, Kinds / Types, Predicate / Arc, Object / Resource);
* Upper / Matching (Business)
* Units of Measurement (continuos) APIs / Ontology.
* Discrete (events) APIs / Ontology.
* Map. Reduce
* CAN / CAM RDF
* FCA Lattices
* Data, Schema, Behavior Layers Augmentation Statements shifts Quads Statements Resources from more specific to more upper and general ontology Contexts Roles (primitives). Aggregation.
* Reify Contexts: Statements / Mappings / Transforms, Kinds, Resources. Templates Matching.
* Inference / Augmentation Templates: Dataflow (Roles fixtures).
* Upper / Matching (Business)
* Units of Measurement (continuos) APIs /  Ontology.
* Discrete (events) APIs / Ontology.
* Encodings: Ternary. FCA. Graphs. Primes / Bitstrings. Tensors.
* Augmentations / Transforms:
* Data matching. Resource equivalence: identity / comparisons / order transforms.
* Schema matching. Predicates equivalence. Domain / Range types. Order: data flow contexts.
* Behavior matching. Domain / Range values applied functional predicates identity. Order: data flow interactions.
* Data Matching Services (Nodes / Resources)
* MVC: Model; Index Service
* DCI: Data; Index Service;
* Schema Matching Services (Kinds / Types)
* MVC: View (flows / prompts); Registry Service;
* DCI: Context; Registry Service;
* Behavior Matching Services (Predicates / Arcs)
* MVC: Controller; Naming Service;
* DCI: Interaction; Naming Service;
* Augmentations:
* Alignment (Data Matching)
* Activation (Schema Matching)
* Aggregation (Behavior Matching)
* Index Service
* Naming Service
* Registry Service
* DCI Relationship Contexts: Data / Schema / Behavior Model. DCI / MVC / Relationships Upper onto matching: gestures / flows.
* Metaclasses: PredicateKind SubjectKinds / ObjectKinds.
* Relationship: (Relationship, SubjectKind, PredicateKind, ObjectKind);
* PredicateKind of SK / OK. Employment(Employer, Employee); Employment (Employee, Position);
* Relation : (Relationship, Statements / Context, Role, Occurrence);
* Role : (Relation, Resource, Occurrence, Metaclass : Kinds);
* Occurrence : (Role, Relation, Context / Relation Statements, Resource);
* Discrete N-ary Relationship Aggregated Statements:
* Aggregated Statements traversal: expanded SPO form.
* Context: (Relationship : Predicate Kind, Relation : Statements, Role : Kind, Player : Resource);
* Predicate Kind of Reified S SK, O OK. (Relationship: Employment, Roles: Employee SK, Employer OK). Employment PK aggregated by Subjects and Objects Kinds. Relation Statements: Aggregated SK, PK, OK by Contexts Statement Kinds.
* (Working, workingRelationStmt, employer, IBM);
* (Working, workingRelationStmt, employee, John);
* Dimensional Relationships:
* (Dimension : Relationship, Measure : Relation, Unit : Kind, Value : Resource);
* (Time, oneHourStmt, minutes, 60);
* Distance Dimension: PK of Time SK / Meters OK. Define Dimension in terms of Relationship Kinds.
* Dimension Measure Statements: Domain PK Statements. Kind interface for Functional Transforms.
* Unit: PK Measure SK / OK Statement Kinds (SK / OK Members).
* Value: Dimension Measure Statement Kind Resource.
* Dimensional Order / Comparison: OrderKinds. Templates (populate).
* Order: Comparisons.
* Resource Hierarchy Categories: Resources, Kinds, Statements, Mappings, Transforms.
* Resources:
* C: Contexts (Transforms / Class), S (Occurring Resource), P (Mapping / Transform), O (Occurrence Resource) Resources.
* Kinds:
* Kinds: SubjectKind, PredicateKind, ObjectKind. Peter valueOf Employee Kind Category Instance.
* Kinds (Wrapped Types / Class). Encoded in Statement Context (Transform / Class). Resolvable Resource Types: Subject SubjectKind (Subject POs) Kind Statement Context.
* Augment Resources with Kinds in Context. Core Model Transforms Mappings Instances / Roles: Kinds and Singleton (Resource) Class.
* Statements:
* Statements : Performed / Matching Contexts Transforms.
* Statements Resources: C: Transform / Class, S (Occurring Resource), P (Mapping / Transform) /  O (Occurrence Resource). S/O. Values: Resources / Reified
* Values. Resources Model / Domains Objects Hierarchy. Activation Augmented Types. Alignment Domain APIs (Measures, Dimensions, etc.).
* Mappings:
* Statement Predicates.
* Context / Statement Transforms / Mappings:
* Dual Occurrence of Transform (Mapping) / Occurring of Mapping (Transform).
* Transforms:
* Context / Transform / Class: (Context, SK, PK, OK). Performable Mappings Templates from learnt input / inferred Statements.
* Reification: Augmented Model.
* Encoding: Representations: Instances / Literals Encoding. URNs. Resolution: sameAs Mappings / Parsing. Occurrence / Occurring domainOf / rangeOf Type Inference.
* Occurrence / Mapping Declaration: (Mapping / Class / Metaclass, Resource / Instance, Transform / Occurrence / Context / Statement, Resource / Instance / Role);
* Occurrence Object Member of Subject as Transform / Function Role.
* Occurring / Transform Application: (Transform / Class / Metaclass, Resource / Instance, Mapping / Occurring / Context / Statement, Resource / Instance / Role);
* Occurring Subject Member of Object as Mapping / Function Role.
* Aggregation Statements Types:,
* (Kinds / Types, Object / Resource, Predicate / Arc, Object / Resource);
* (Object / Resource, Kinds / Types, Predicate / Arc, Object / Resource);
* Aggregation Statements Types:
* (Kinds / Types, Object / Resource / Predicate / Arc, Object / Resource);
* (Subject / Context, Kinds / Types, Predicate / Arc, Object / Resource);
* Reification. Infer Kinds / Types vía repeated matching of Statements Attributes occurrences. Quads Encoded. Attribute relation: Mapping from and edge of whim an Arc / Predicate is source.
* Encoding: Reification of Statement types to canonical form:
* (Context, Subject, Attribute, Value);
* Aggregate / de Aggregate mappings expansions.
* Operations: Monads, Transforms, Dataflow. Templates / Mappings for processing representations. TODO

### Persistence Sail

* Apache Kafka: Event Sourcing. DLT.
* W3C DIDs: Distributed Identifiers
* Connectors: Event Driven Dataflow.

## Encoding Layers Object Model

### Serialization

* Type Inference: Kinds (Classes):
* Aggregate same Attributes occurrences for sets of Resources sharing same Attributes. Activate Context Transforms Kinds. Activate Kinds Resources Statements.
* Meta Model Roles: Resource (CSPO), Kinds (SK, PK, OK, Contexts), Contexts (Statement, Mapping, Transform), metaclass, class, instance, context, role, occurrence, attribute, value. Inputs URN, Sets, DOM Model Alignment.
* Encodings: Ternary. FCA. Graphs. Primes / Bitstrings. Tensors.
* Layers Augmentation. Data, Schema, Behavior Layer wise Statement Roles.
* Alignment: Data Matching Augmentation.
* Activation: Schema Matching Augmentation.
* Aggregation: Behavior Matching Augmentation.
* DagCBOR. XML (Beans, Externalizable): Functional Roles (metaclass, etc) Functional Transforms (Aggregation Templates). HAL / JSON: Functional Fields (codat / data flow).
* P2P Browser: Addressing, linking and annotation / embedding. Items: subjects occurrences in roles in contexts: Documents / resources semantic linking. Workflow session abstraction of resources in roles in learned / inferred domains (applications). Content types, addressing, representation (i.e.: mail / chat conversation URI addressed / linked as project / document cause role, person in picture occurrence in marriage event role, etc.). P2P Browser rendering declarative state flows / contexts / interactions embeddings addressable / embedded resources in roles in contexts.
* Search, find and browse semantically. Copy and paste any meaningfully labeled addressed resource into any other context resource. Discover intelligent insights from linked knowledge bases rendered together with the resources that originated them while navigating with an enhanced browser which allows for conceptual based relations and dimensional reasoning traversal.
* Connect any addressable content type representation into document (semantic resource) embeddings playing roles: purchase, invoice. Resource annotations. Addressing and type / representation handlers as browser plugins. Custom protocol adapters (example: mail, chat conversations playing role in interactions; picture: scanned receipt). Meta browser involving session workflows. Annotations (subject / occurrence: picture / event; paragraph mentions interview: semantically augmented link to audio / video of subject).
* Basic inference (resource augmentation: activation, aggregation, alignment models).
* Class (type) / ID (instance) inference.
* Attributes / Links inference.
* Context / Role inference.
* Message driven reactive P2P architecture. Synchronization. Event sourcing (distributed consistency).
* Implementation: Health Care domain. Functionalities such as kinds of ERP, CRM, SCM (B2B / B2C) for health: HMS (Health Management System) The Browser provided will act as an overlay (of integrated resources). Example linked resources (records): symptoms, measures, interviews (previous QA forms), analysis / trials, results, diagnoses. Clinical history. Integration: peer / node, cloud. Metamodels. BI / Big Data. Rules. Planner. Business Process Management. Resources allocation / logistics (optimize resources allocation). Workflows (ontology learnt).
* Augmentations:
* Data Match Model (Sets).
* Infer Kinds / Flows.
* Models:
* (Statement, Kind, Attribute, Value);
* (Template, Statement, Kind, Attribute);
* Schema Match Model (Graph).
* Infer Types / Hierarchies / Shapes.
* (Statement, Resource, Kind, Attribute);
* (Template, Statement, Resource, Kind);
* Roles / Behavior Match Model. (Roles). Infer Contexts / Interactions / Measures.
* (Measure, Dimension, Unit, Value);
* (Resource, Measure, Dimension, Unit);
* (Statement, Resource, Resource, Unit);
* (Template, Context, Statement, Resource);
* (Roles with original wrapped Values).
* Template Transforms:
* Subjects: (Subject, Occurrence : SubjectKind, Attribute : Resource P, Value : Resource O);
* Predicates: (Predicate, Occurrence: PredicateKind, Attribute : Resource P, Value : Resource O);
* Objects: (Object, Occurrence : ObjectKind, Attribute : Resource P,  Value : Resource S);
* SubjectKind (SK): Predicate / Object Intersection. Occurrence:
* (Context : SubjectKind, Occurrence : Statement, Attribute : Predicate, Value : Object);
* PredicateKind (PK): Subject / Object intersection:
* (Context : PredicateKind, Occurrence : Statement, Attribute : Subject, Value : Object);
* ObjectKind (OK): Predicate / Subject intersection. Occurring.
* (Context : ObjectKind, Occurrence : Statement, Attribute : Subject, Value : Predicate);
* Statement / Mapping / Transform: Subject / Predicate / Object intersection:
* (Context : Transform, Occurrence : Mapping, Attribute : Resource T, Value : Resource : U);
* (Context : Mapping, Occurrence : Transform, Attribute : Resource T, Value : Resource U);
* (Context : Statement, Occurrence : Mapping, Attribute : Resource, Value : Resource);
* Transform::Mapping::Statement::Kind::Resource;
* Resource::Kind::Statement::Mapping::Transform;
* Aggregation: Statements for each Context Occurrence Attribute / Value. Mapping: for each matching Attribute / Value apply Transform, render Statement. Context: Transform / Class. Occurrence: Subject. Normal Form.
* Aggregation: Statements for each Context Occurrence Attribute / Value. Occurrence: Subject. Normal Form.
* Encoding / Matching:
* Functional Primitives: (Matching). Graph Shapes Model. Layers. Example: reify / render / match "uncle" relation / "marriage" situation from graph statements.
* Canonical Statement:
* (C, S, P, O);
* Canonical Type Statement
* (Type, Resource, Attribute, Value);
* Canonical Subject Statement
* (Resource, Type, Attribute, Value);
* Type centric Statement (Dimensional)
* Subject centric Statement (Discrete)
* Reification: Meta Models, Statement Layers Roles
* Layers: Data, Schema, Behavior
* Upper Hierarchies alignment of Metaclass, Class, Instances, Contexts, Roles, Occurrences in Layers Contexts Statements. Meta Model Primitives: Layers Templates.
* Encoding: primes / primitives / metaclass, class, instance characteristics. Numeric systems / aggregation, groupings, comparable rels. Contexts: quads / tensors (recurs.). Embeddings. State (events / attributes /flows): aggregate kinds / processes / actions / dimensional transitions (data / schema / behavior types / templates).
* Framework: Dimensional / hypermedia application semantic modelling (HyTime / Topic Maps / SW). Abstract Metamodel layers (dimensional data, schema and behavior: templates). Rendering (services, apps: XUL / ZUL, SVG). XML XSL / XPath / XLink / XPointer / XQuery metadata. Plug backend layers (declarative templates / microservices).
* Sets, groups, categories, graphs (formalization / DM, RM).
* Schema, microformats (annotations / metadata), JSON-LD. REST HATEOAS / HAL from Metamodels (resources, rel / declarative discovery links, self links: lambda / runat messages semantics). Declarative descriptions / locators (discovery / signatures) frameworks.
* HyTime (dimensional addressing, traversal: gestures / events), TopicMaps, XML, XSLT, XPath, XLink, XPointer, XQuery. Discovery / signatures (arcs / nodes) / query data / node paths (XPath encodings). External metadata linkbase arcs (pointers) actuate traversals, referrer headers, HAL / HATEOAS 'self' rels. Graph aggregations (XLink / metamodels).
* App (proof of concept): expenses, organizer (soccer / medical appointments / generic tasks assistant / wizard, objectives). Shopping use cases: actors (customer / seller), supply chain. Campaigns (selling / buying objectives / flows).
* Formalization: metamodels / alignments (dimensional / layers) data / reference models (parsing). Canonical forms (graphs iso / rels). Quads (self occurs loops / recursive). Merge / alignments (id, attrs, roles: encode rels):  kinds / attrs / values. ML (graphs).
* Serialization / externalization. Serving models. Data URL scheme.
* Deploy: lambda, runat, serverless.
* Monads. Parsers. Field access interfaces.
* ANN / Deep pipelines / flows. Reactive streams.
* Embeddings (words). NLP.
* Statements 'ordering' (streams / events sourcing: declarative entities descriptions). Dimensions. Tasks / provenance: steps, states (behavior rels / templates).
* Big Data, SemWeb, ML, Graphs.
* ID Alignment: metaclass / class / instance kinds (attributes / values), occurrences graphs analysis from primitives to objects lattices (rels / roles).
* Metamodel (labeled arcs: occurrences).
* Bus address encodes input / output 'signatures' (registering / discovering functional 'domains': backends, services, predictions / activations). Location (expressions) resolvers.
* Bus address encoding: dimensional data, schema, behavior inputs / outputs (message) signatures. Locators: metamodel addressing expressions / parameters. Invocations: Locator message instances (knowledge domains functional hypermedia / browseable messages APIs).
* Data: price, information: price var, knowledge: price tendency. Dimensional data, schema, behavior. Layers.
* Stanbol, Any23, Metamodel, Teiid. OData. HAL. GraphQL. TMDM / TMRM / TMCL (Constraints / schema as data: Validating RDF Data, metamodel data / reference models).
* Encoding: Translate graph into dot notation keys / values property chains. Arrays / templates / variables / placeholders.. Indexing / search / alignments.
* Property graph encoding implementation. Metamodel, layers, persistence (index, registry, naming) search / algorithms. Instance, class, metaclass. Alignments.
* key1 = context.subject.predicate.object.attribute: '1234';
* key2 = key1.context.attribute: 'abc';
* Paths: [index, occurrence, context] expressions / resolutions.
* Schema: annotation kinds:
* Entity / ID.
* Class ID (ID Type: genre : ID, ID Value: male : Entity).
* Instance ID (ID Type: SSN : Entity, ID Value: 1234 : ID)
* Attrs / Rels (Class ID, Instance ID) pair.
* Contexts / Roles (Attrs, Rels pairs: Class ID, Instance ID context / role). Parent / child hierarchies.
* Schema: annotation(annotation\_id, annotation\_kind, annotation\_subject);
* Schema: class\_id\_type(class\_id\_type\_id, name); parent\_class\_id\_value\_id: two tables.
* Schema: class\_id\_value(class\_id\_value\_id, class\_id\_type\_id, value, annotation\_id, metadata);
* Schema: instance\_id\_type(instance\_id\_type\_id, class\_id\_value\_id, name);
* Schema: instance\_id\_value(instance\_id\_value\_id, instance\_id\_type\_id, value, annotation\_id, metadata);
* Schema: annotation\_subject(subject\_id, type\_id, value\_id, name); class / instance browse / refinements.
* Schema: metadata, class / instance types values JSON forms.
* Infer function / models from inputs / outputs.

### Model

* Meta Model Roles: Resource (CSPO), Kinds (SK, PK, OK, Contexts), Contexts (Statement, Mapping, Transform), metaclass, class, instance, context, occurrence, attribute, value. Inputs URN, Sets, DOM Model Alignment.
* Core Statements Roles Resource Shapes:
* Occurrence: (Class, Instance, Attribute, Value);
* Occurring: (Class, Instance, Occurrence,  Role);
* Context Statements / Mappings / Transforms Relations: order, equivalence, roles, etc.
* Resource of Resource Monad: Occurrence  / Occurrings Quads CSPO Members Aggregation Transforms.
* Core Statements Roles:
* Occurrence: (Class, Instance, Attribute, Value);
* Occurring: (Class, Instance, Occurrence,  Role);

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

ResourceURN aggregates Resource Occurrences Kind. Encodings.

KindURN : (ResourceURN, Kind, Occurrence, Resource);

ResourceURN aggregates Kind Occurrences Resources. Encodings.

OccurrenceURN : (ResourceURN, Occurrence, Kind, Resource);

ResourceURN aggregates Occurrences Kinds Resources. Encodings.

SubjectResourceURN : (ResourceURN, Resource, Occurrence, SubjectKind);

Functional APIs:

Resource::getOccurrences

Resource::getKinds

Occurrence::getResources

Occurrence::getKinds

Kind::getResources

Kind::getOccurrences

Encoding:

IDs: Resource, Occurrence, Kind. ResourceURNs aggregation / order encoding.

Graph / Tree List Parent / Child hierarchical encoding / hashing.

Functional Data Flow. ResourceURN Events. Message Logs Streams / Traversal (Index Persistence Events Graph Interfaces):

(Occurrence, Kind, Resource)

(Occurrence, Resource, Kind)

(Kind, Occurrence, Resource)

(Kind, Resource, Occurrence)

(Resource, Occurrence, Kind)

(Resource, Kind, Occurrence)

Quads / SPOs hierarchical list encoding.

(C (S (P (O).

Dataflow (value expressions). Signatures (events subscriptions: domain / range). Encode Order.

Hashing: IResourceURN, IOccurrence, IKind, IResource. Nested recursive URNs aggregations. Order / Mappings / Traversal.

Input / Canonical: Match Interfaces / Signatures: (Context, Occurrence, Attribute, Value); Attribute / Value Roles in matching interface context. Order / hierarchy encoding / functions (sorted wrapped functional collections: wrappers set comparators / aggregation axis).

* Input Layers: Core Statements Roles Resource Shapes: (interleaving graphs models) : Statements CSPO.
* Occurrence Input: (Class, Instance, Attribute, Value);
* Occurring Input: (Class, Instance, Occurrence, Role);
* Output Layer: (Template, Context, Statement, Resource); Input OPSC Resources wrapped in output Layer Roles.
* Sets, Graph, Roles: Services / Augmentations helper Models / Facades.
* Example:
* (Amantes, Pedro, Ama, María);
* (MariaAmadaPor, Amantes, Pedro, Ama);
* (Amar, MariaAmadaPor, Amantes, Pedro);
* (PedroAmaA, Amar, MariaAmadaPor, Amantes);
* (Amor, PedroAmaA, Amar, MariaAmadaPor);
* Meta Model Roles: Resource (CSPO), Kinds (SK, PK, OK, Contexts), Contexts (Statement, Mapping, Transform), metaclass, class, instance, context, role, occurrence, attribute, value.
* Core Statements Roles:
* Occurrence: (Class, Instance, Attribute, Value);
* Occurring: (Class, Instance, Occurrence,  Role);
* To Do.
* Encodings: Ternary. FCA. Graphs. Primes / Bitstrings. Tensors.
* Layers Augmentation. Data, Schema, Behavior Layer wise Statement Roles.
* Alignment: Data Matching Augmentation.
* Activation: Schema Matching Augmentation.
* Aggregation: Behavior Matching Augmentation.

### Runtime

* Models:
* Activation: (Activation, Statement, Resource, RoleKind);
* Aggregation: (Aggregation, Activation, Form, Template); Form, Template: statements, kinds / resources of Activation RoleKind occurrences.
* Routes: (Route, Aggregation, Edge: Activation, Node: Form);
* Models aggregation metadata (describes models / models values):
* Activation Kinds (schema / dims classification : metamodel layers): (Kind / Resource : unit, super Kind : dimension, Activation : measure, Kind : value).
* Aggregation layers (context / role regression): (Template : player, Template : occurring layer, Aggregation, Template : occurrence value).
* Paths attributes (attributes augment / clustering): (Node : player, Node : occurrence / context, Route : attribute, Node : value).
* Template / Form (Kinds) upper root hierarchy: Entity, Kind, Class, Flow, Behavior.
* Message wrapper (functional / monads / elements events callbacks / type system / streams / signatures / headers). Exchange / Protocol.
* Model declarative Aggregations:
* Entity, Kind, Class, Flow, Behavior : Facts.
* Inferred models:
* Dimension, Unit, Measure, Value : Schema.
* Context, Concept, Sign, Object : DCI / App.
* Context, Occurrence, Attribute, Value : Grammar
* Alignments:
* Classification (class / ID).
* Regression (role in context)
* Clustering (links / attributes).
* Models: trained / augmented on each Message exchange (online training). Events: Processor IO implementation.
* Models: Collections (observables) of Activations, Aggregation, Routes. Perform inferences. Parameterized Messages.
* Map Resource, Statement / Template / Forms, Node / Edges to objects / tensors (events).
* Reactive API (Message streams : Protocol) dataflow / IO:
* Activation (in quads) - Aggregation (eg.: Entity / Class) - Routes (augment attributes) - Aggregation (e.g.: Class / Entity) - Activation (out quads).
* Protocol:
* Upwards: Statement::toLayer(model);
* Downwards: Statement::fromLayer(model);
* Message pattern builder (Activation). Message as (observable) resource “persists” as an endpoint stream till yield results (services implementation).
* Message Exchange functional transforms:
* Order
* Comparison
* Iteration
* Jumps
* Backend connectors (Resource / Statement Layers / IO : Exchanges):
* RDF / Graph DB (event driven) Backend connector.
* Reactive (Vert.x) Backend connector.
* Services: Spring / OSGi (events) Backend connector.
* TensorFlow (events / dataflow) Backend connector.
* RDBMS, Messaging, Services Backend connectors.
* Feature columns: FCA lattice bitstring.
* Deployment:
* Ont DIDs: Encoding. Addressing, Routing. Distributed logs / routing layer (Alignments : Transactions / Exchanges Messages logs). Endpoint client bindings Processor (resolve service node address / message pub / sub: streams / events logs). Business (Metamodel) layer message abstractions (IO: import / export converters, DCI / DOM / object mappers / content types handlers).
* Services layer. Nodes Backend. Message routes / protocol resolution (based on DIDs addresses / headers / logs).
* Nodes Runtime. Business (Metamodel objects) layer. Data, information, knowledge models (aggregated). Resource / Statement : Template / Form. Factory / Builder (for each model): Forms (of layers) from URLs / objects factories. Layer facade: Aggregate models / perform alignments / de-aggregate models (in / out Message Dataflow).
* Dataflow: Models (layers) observing each other, Resources observing occurrence Statements, Statements observing occurring Resources. Forms CSPOs specialization (pre / post aggregation / alignments).
* Forms CSPOs specialization: retrieve / consolidate / determine equivalent CSPOs, Kinds between Messages. Resolve common entities in context. Aggregation / inference facade.
* Activation: determine kinds / resources from equivalent IDs. Ontology matching. Classification.
* Aggregation: determine declared aggregation layers occurrence.  Determine declared Aggregations (Entity, Kind, Class, Flow, Behavior). Dims. Schema. Clustering.
* Paths:  determine role in context of occurrence in path. Regression.
* Retrieve requested Form hierarchy from performed alignments (layers). Build response graph.
* Apache Jena / [rdflib.js](http://rdflib.js): RDF Store Backend. Model persistence layer. Persists Business layer inputs / inference layer outputs Metamodels.
* TensorFlow / Aggregations / Alignments inference: Nodes Engine. Model inference layer.
* Layers: async event driven interfaces (Exchange contexts: Messages). IO flows Processors.
* Activation (Kinds) / Aggregation (Upper) / Alignments. In / out IO. ResourceManager feeds Resource / Statement. Events handlers dataflow. Upper ‘kinds’: roles.
* ResourceManager : Observer, Observable. CRUD source / dest notifications / sync. Lazy fetch (query). Manager : Resource having occurrences / occurring in Statements.
* Main / upper hierarchy:
* Kind : Entity : Statement : Resource.
* Behavior : Flow : Class : Kind.
* Resource events:
* onContextOccurrence
* onSubjectOccurrence
* onAttributeOccurrence
* onValueOccurrence
* Statement events:
* onEntityOccurrence
* Entity events:
* onKindOccurrence
* Kind events:
* onClassOccurrence
* Class events:
* onFlowOccurrence
* Flow events:
* onBehaviorOccurrence
* Bound functions (getters / setters CSPO / Kinds):
* Class / Interface API (config deployment and application instance graphs declaratively in model statements / hierarchy): Peer / Node, Model, Graph, etc. (Resources). Observe ResourceManager, augments on new Resource, notifies ResourceManager on event updates with augmented data. Dataflow bindings on configuration statements.
* RESTFul API Resource endpoints for native RDF application configuration (backends). Events handlers. Example: RDBMSResourceManager.
* Resource (upper hierarchy) : URL.
* Resource<URL>. Events: HTTP verbs (Processor) bound functions. Navigate representations (body / content type "activation" bound functions) / verbs. Headers. Ont DIDs, endpoints (uniform API)a. Resource (onOccurrences) : bound functions.
* Resource.contexts / onContext / getContext (fMap): Stream<Statement>;
* Resource.subjects / onSubject / getSubject (fMap): Stream<Statement>;
* Resource.attributes / onAttribute / getAttribute (fMap) : Stream<Statement>;
* Resource.values / onValue / getValue (fMap) : Stream<Statement>;
* Resource.kind(Statement) : Kind;
* Statement.kind(Resource) : Kind;
* Resource.occurrences(Resource) : Statement (hier parent / child; example: Entity in Kind).
* Resource.occurring(Resource) : Statement (hier child / parent; example: Kind Entity).
* Upper ontology aggregation: monadic wrapper for lower level contexts. Example: Behavior over Flow augmentation. Resource.fMap(Resource) and events / streams logic.
* Promise (async) based pipelines for layers augmentations.
* Augmentation: Activation, Aggregation, Alignment. According Resource URI content type mappings. Registered processors. (providers extract Statement from representations). Example: Image / Face (attributes / links, coords qualified); Face / Gender (classification); Face / Subject (ID); Subject / Occurrence (context / role).
* Content types: image/png;face, text/xml;faceImgCoords, text/rdf;faceImgPoints
* Dimensional: comparisons / distance. Nested previous / current / next measures (next hour, next minute). Contexts: next distance at next time at current speed, next day at day of week, etc.
* Serialize: Distance, Previous, Next (Value)
* (Value, Distance, Previous, Next);
* (Measure, Value, Distance, Previous);
* (Unit, Measure, Value, Distance);
* (Dimension, Unit, Measure, Value);
* (Concept, Dimension, Unit, Measure);
* Encode layered statements orderings: dimensional / upper ontologies. Kinds / Roles. Flows / Templates / Forms. Previous, current, next (octal value temporal order feature?).
* Primitive values: self, the, this, that, inside (here / that), outside (here / that), after, before, during, etc. Value variables / placeholders / wildcards.
* Assert class hierarchies, containment relation, order relation, roles relation (occurrences / metaclass).
* Unify (link) upper / dimensional ontologies.
* Upper: (Behavior, Flow, Class, Kind); Entity
* Dimensional: (Concept, Dimension, Unit, Measure); Value : Entity
* Type, State.
* Event : Measure, Kind.
* State: Concept, Behavior.
* Axis: Dimension, Flow.
* Type: Unit, Class.
* Event: Measure, Kind.
* Grammar / Type System:
* (Event, Event, Event, Event);
* (Type, Event, Event, Event);
* (Axis, Type, Event, Event);
* (State, Axis, Type, Event);
* (Object, State, Axis, Type);
* Primitives: disabled: true / enabled: false. Complements / supplements in contexts.
* Sets, categories, functional models (reference / data: upper ontologies common abstractions). Formalization.
* Messages (Events): Declarative transforms specifications (capabilities) / instances (exchanges). Activation, Aggregation, Alignment declarations.
* Message<Resource<? : URI>, R : URI> : Applicable;
* Visitor: Message::apply(Resource<?>) : Message<Resource<R>>; Reactive Dataflow.
* Upper ontologies classes instantiated as Resources (Concept, Behavior, Object hierarchies : reify / reflection).
* Metamodel ontologies
* Contexts / Interactions:
* (Entity, Statement, Resource, Resource);
* (Kind, Entity, Statement, Resource);
* (Class, Kind, Entity, Statement);
* (Flow, Class, Kind, Entity);
* (Behavior, Flow, Class, Kind);
* Dimensional:
* (Value, Distance, Previous, Next);
* (Measure, Value, Distance, Previous);
* (Unit, Measure, Value, Distance);
* (Dimension, Unit, Measure, Value);
* (Concept, Dimension, Unit, Measure);
* Grammar / Type System:
* (Event, Event, Event, Event);
* (Type, Event, Event, Event);
* (Axis, Type, Event, Event);
* (State, Axis, Type, Event);
* (Object, State, Axis, Type);
* State: Concept, Behavior.
* Axis: Dimension, Flow.
* Type: Unit, Class.
* Event: Measure, Kind.
* Formalization (class / metaclass / instance):
* (Context, Occurrence, Attribute, Value);
* (Context, Sign, Concept, Object);
* Sets, categories, models (reference / data).
* Transforms: translate source resources (observable URIs) into metamodel ontologies. Activation, Aggregation, Alignment.
* Transform, Message / DB Resource:
* (Table, PK, Column, Value);
* Message<<Resource<URI : DBURI>>, R : Statement>
* Applicable: CRUD / Query Messages.
* Transform, Message / Hierarchical (JSON / XML Resource):
* (Context, Parent, Value, Children);
* Message<<Resource<URI : EndpointURI>>, R : Statement>
* Applicable: HAL / REST Messages.
* Transforms: translate source resources (URIs) into metamodel ontologies. Augmentation: Activation, Aggregation, Alignment.
* Activation, Aggregation, Alignment : Message specs (reified ontologies / metamodel / augmentation resources reified "verbs"). Reactive dataflow.
* Message<<Resource<URI>>, R : Statement>; R: Activation, Aggregation, Alignment (Augmentations).
* Messages dataflow order: inputs, outputs (static / runtime) type flows. Comparison (dispatch) by specialization hierarchies, super: first, sub: next (order in contexts traversal : execution trees / graphs “definitions”, metamodel “activated”. Runtime content type flows.
* Quad Statement Equation Encoding:
* aX⁴ + bY³ + cZ² = dW;
* a, b, c, d: classes (SPOC).
* W, X, Y, Z: instances.
* Powers: statement SPOC roles.
* Terms: occurrences (values).
* X (object) is Y (predicate) for Z (subject) in W (context).
* Operate (filter / sort / map) over features. Satisfy context dW (calculated desired results class / instances) SPOs. Messages / dataflow: aggregate possible mapping / filter verbs / results into reified resources. Alignments.
* Identifiers: embedded features (substituted coefficients and variables).
* Normalize Terms (SPOs) class / instance according context (dW) class / instance. Normalize dW Term.
* Alignments: Class / ID, Attributes, Roles (order, containment, other relations) verb Messages. Message of possible context input / desired output contexts.
* Nested contexts: ontologies hierarchies.
* Nested contexts: ontologies hierarchies. Message aggregation: composite contexts: Message verbs, predicates, subjects, objects composition (Message as template for context resolutions). Term class equals Term variable: class Term.
* Contexts in model hierarchies (behavior, dimensional, grammar): aggregate / map class / instance values on Activation / Aggregation (map URIs / IDs: DIDs, endpoints).
* Encoding: quad statement.
* Class Terms: statement templates.
* Resolution via execution graph dataflow. Alignments (encoding as verbs: resources / messages):
* Class / ID: Common upper / same (equivalent) contexts results.
* Attributes: Terms applying. Predicates satisfying quad encodings results.
* Roles: Role in contexts. Context as object Term in predicate role for subject context satisfying enclosing context statement results.
* Comparison: Sort alignments execution graph dataflow according context / verb resources / messages order. Iterations. Jumps. Comparison contexts (alignments): Class / ID, Roles, Attributes. Match next contexts. Nested contexts.
* Augmentation: Activation, Aggregation, Alignment declarative statements (resources, messages / verbs: resource application, reified metamodels). Statements, Terms, Class Terms. Variables, placeholders.
* Message<T, R>;
* T, R: Resource<URI>;
* Apply resources: T : Person (class), R : Employee (kind) : Employees (kinds, occurrences).
* Apply resources: T : Employee (kind), R : Person (class) : Person occurrences as Employee (classes, occurrences).
* Metamodel graphs. Dynamic activation, aggregation, alignment (execution graph dataflow) resource declarations.
* Execution (dataflow) graph: Messages (transforms) resource declarations (activations, aggregations, alignments). Member (attributes) access. Graph statements describe graphs available interactions (messages / transforms). Interactions invoked according ontology alignment rules / metamodel aggregation (order / contexts). Entailed results appended (activated, aggregated, aligned) into originating models.
* Ontology alignment rule: (Occurrence, Context, Class, Metaclass); Mappings (message instances / transforms) for relations between metaclass / class / instance.
* Member access: Mappings between Context : T and Context : R (URI member specialized) resources.
* Augmentation: Activation, Aggregation, Alignment described as dataflow execution message flows (reify message resources in metamodels).
* Execution graph:
* (Message, T : LHS, R : RHS, MappingValue : R);
* (Interaction, Message, T : LHS, R : RHS);
* (Context, Interaction, Message, T : LHS);
* (Dataflow, Context, Interaction, Message);
* Flow ordering by Message inputs / outputs. Activate, Aggregate, Align: learn dataflows.
* Reference model: model abstractions used to describe models defined itself in terms of (a) model(s).
* Mapping to / from metamodels / model instances representations. Ontology, Inference, Alignment, Constraints, Shapes. Encodings.
* Subjects / Contexts: Occurrences / Roles.
* Occurrences / Subjects: Contexts / Roles.
* Subjects / Roles: Contexts / Occurrences.
* Attributes / Values: from entities types applicable for each term (example: occurrence attrs / vals in corresponding ctxs).
* Metaclass, Class, Instance.
* Data: items / attributes / values. Classification, types (activation).
* Information: Data items in context. Roles. Items relations (aggregation).
* Knowledge: Information evolution. Predictions (alignments).
* 1) XML: XSL / XPath / XLink / XPointer / XQuery.
* 2) Semantic techs. ISO (upper, data / reference models). KR. Logic. Rules. Inference. Graphs (dataflow). Tagging (annotations metadata): Class / ID, Links / attrs, Context / roles of subjects / occurrences (Content type activation / annotation: document describing picture occurrence with annotations in context, document / picture role: diagnose / evidence, prescription / documentation, invoice / originating sales order, profile picture / holidays picture. Roles in contexts activation "flows": orders, invoices, etc.). ERP upper onto. Dataflow (operations: flow transitions / links, addressing according states).
* Linked and addressed "activated" content (subjects / occurrences). Profile: profile picture, contacts, messages; Holidays: holiday's picture, place, itinerary. Identify same picture in both contexts. Infer rels.
* 1) & 2) plus addressing / resolution:
* Translation / Activation examples: link pictures with DB records (in context / roles / occurrence, attributes of subjects). Identity, Attributes, Roles Alignments.
* Expose, consume (endpoint / service, datasource / backend): same interfaces (dual behavior). Observer. Service resource request / response to events, datasource reacts / generates events. Functional dataflow / models (activation bindings).
* Grammars: syntax / semantics / pragmatics. Statements: attribute / logic grammars. Logic, rules embedded.
* Semiotic (syntax / grammar: data, semantic: dimensional, pragmatic: behavior) inferences (activation / grammars). Models (data / metamodel, schema / dimensional, behavior / application) productions / rules (Resource IDs Graph / Activation statements). Parsing (dataflow graph states).
* Dimensional: encode relations, conditionals, flows (containment, space, temporal, causal, states, part, whole, roles, etc.) via orderings and ID Graph statements.
* Each node has internal networks / graphs and exposes its graphs / models through DIDs. Networks nodes uses DIDs internally to retrieve Resource features. Services. (TensorFlow / Vert.x / Ont DIDs).
* Dataflow: Resources / Activations graphs / networks. Declarative IDs graph over data / schema / behavior graph models. Activation: apply Resource in role (operation) in statement (context) according IDs graph dataflow activation flows (production rules). CAM / declarative transforms specifications. Order relations encoding (models / IDs graph).
* Domains. Models (data : Metamodel / schema : Dimensional / behavior : Application). Translation. IDs Graphs.
* Order relation encoding: previous / current / next statements (primitives comparator ternary operation: expected next, self, the, this, that values assigned to roles in contexts / features graphs). Cycles (digit positions, nested comparators).
* Order: Layers Statement / Contexts comparison (Dimensional). Superclass: previous. Subclass: next.
* Octal encoding: comparison result (CSPO Statement).
* Graph: encode computation / control structures. Apply transforms (ops) to ordered / filtered streams. Control flow: next declarative resource / operation dataflow (transform results activation).
* Turing complete: states (initial / halt). Read: eval state / input, action (write, jump). Conditional branching / iterations.
* CAM: content / adresses, data. Index. Ternary X: don't care. ALU (PC). Comparator.
* Monads. Lang parser.
* Activation: Map / Reduce: graphs / activation. Filter / sort (functional map apply op over a functor : Template). Reduce / summarize / aggregate according hierarchy traversal combination operation (fold hierarchically CSPOs : Form).
* Map / Reduce: activatable behaviors. Dataflow Activation nodes.
* Control dataflow: activatable Activations.
* Activation: Map (apply) Resource over Resource : Resource. Grammar possible production statements from common hierarchical roles between SPOs of operands. Reduce, fold possible statements by relevance Resource. Possible activations: possible grammar productions given contextualized order of statements in context (dataflow). Being able to extract new knowledge statements from existing ones (Aggregations, ID, Links / Attrs, Ctx Roles Alignments). DIDs transactions: event sourcing features.
* Distributed (aggregation / alignments):
* Map(k1,v1) → list(k2,v2)
* Reduce(k2, list(v2)) → list(v3)
* Resource IDs Graph (metagraph for models bindings: data / schema / behavior Aggregation, Activation, Alignment):
* Resource (cls:id), Statement, Role.
* Statement (cls:id), Resource, Role.
* Role (cls:id), Statement, Resource.
* Models (layered):
* (Context, Occurrence, Attribute, Value); Metamodel. Data.
* (Dimension, Unit, Measure, Value); Dimensional Graph. Assert values for every sub-measure (5min: 4min, 3min, etc.). Express equivalence (60min, 1h). Express context (tomorrow, friday). Schema.
* (Context, Concept, Sign, Object); Application Graph. Behavior.
* Resource occurrences: mode (tense / person / primitives: the, this, that, self, etc) "declination"
* Combinatorial generalization in graph networks.
* Signal processing. Case classes.
* Specification (CSPO + APIs) declarative components (map / reduce / fold event handlers / callbacks).
* Comparison operators (map / filter / fold) : Specification API (Dimensional / Graph contexts).
* Template: Resource Specification.
* Form: Transform Specification.
* Activation (Template: input / Form: binding / Template: output). Capability (interaction) definition. Capability: Activation(s) instances (interactions). Available Message exchanges. Exchange: Message (data) blueprint.
* Activation: query, filter, transform (inference / aggregation / alignments).
* Map / Reduce: activatable behaviors. Dataflow Activation nodes.
* Application Graph:
* Behavior of Resolver / Runtime, Specification / Component, Activation / Capability, Exchange
* Layers:
* (Resource, Occurrence, Attribute, Value);
* (Statement, Resource, Occurrence, Attribute);
* (Entity, Statement, Resource, Occurrence);
* (Kind, Entity, Statement, Resource);
* (Class, Kind, Entity, Statement);
* (Flow, Class, Kind, Entity);
* (Behavior, Flow, Class, Kind);

# Augmentations

* Encoding Layer Model:
* (ResourceURN, Resource, Occurrence, Kind);
* Augmentation Layer Model:
* (Context, Statement, Kind, Resource);
* Aggregation: (C (S (P (O)))) hierarchycal type / value tree addressing. Resources / Kinds Sets Occurrences (Encoding).
* Activation: (Context, Statement, Mapping, Transform) gestures (Aggregations).
* Alignment: Kinds driven type / value Resource matching inference (Activations).
* Services Layer Model:
* (ResourceURN : type, Kind : occurrence, Context : attribute, Context : value);
* Activation:
* Matching / Templates.
* Infer T / U Resources Kinds Wrapped Types / Instances (Place / NY), Infer Transform Class / Metaclass by SPO Kinds. Parse Instances (Subject / Object Resources) Wrapper / Wrapped Types (Kinds Matching). Transform / Kinds Resolve Mapping Statement (noop, merge, add);
* Matching in Occurrence / Occurring Direction.
* Resource::Mapping::Kind
* Alignment:
* Matching / Templates.
* Assert: (Class / Transform, Resource : Kind T, Attribute / Mapping, Resource : Kind U);
* Query: (Class / Mapping, Resource, Attribute / Transform, Resource) : CSPOs;
* Core Model:
* Occurrence / Mapping Declaration: (Mapping / Class / Metaclass,  Resource / Instance : T, Transform / Occurrence / Context / Statement / Class / Metaclass, Resource / Instance / Role : U);
* Occurrence Object Member of Subject as Transform / Function Role. Instance : T has Member Instance : U.
* Occurring / Transform Application: (Transform / Class / Metaclass, Resource / Instance : Kind T, Mapping / Occurring / Context / Statement / Class / Metaclass, Resource / Instance / Role : Kind U);
* Occurring Subject Member of Object as Mapping / Function Role. Instance : T is Member of Instance : U.
* Aggregation:
* Matching / Templates.
* Aggregation (Layers / Sets) Augment / Encode into Core Model:
* Inputs Normal Forms: Dimensional, Discrete, etc. Parse Aggregations into Core Model.
* (Class : Transform, Instance : Kind T, Atribute : Mapping, Value : Kind U);
* (Time, 1h, mins, 60m);
* (Working, 1h, USD, 40);
* (Working, 160h, USD, ?);
* (Employment, anEmployment, employee, John);
* (Employment, employee, Employee);
* (Employment, anEmployment, employer, ABC Inc);
* (Employment, employer, Employer);
* (John: Transform / Singleton, John, employment, anEmployment);
* (Employee, employment, Employment);
* (ABC Inc: Transform / Singleton, ABC Inc, employment, anEmployment);
* (Employer, employment, Employee);
* (John, employmentAt, ABC Inc.);
* (Employee, employmentAt, Employer);
* (ABC Inc, employsFor, John);
* (Employer, employsFor, Employee);
* Model Augmentations / Domain Augmentations. Transforms. Templates. Matching Data Instances (Functor / Mapping) domain / range types / values dataflow data / schema / behavior.
* Templates Statements basic application: between two Statements: given a Template Statement relation URN (i.e.: common superclass, common occurrence context, etc.) Aggregates Aligned Activated Schema results. Nested URNs (context).
* Encoding URN to / from Statements: Dataflow Encoding. Infer Possible Model / Domain Functors / Transforms. Statement Layers, Meta Model Roles: routes / paths (recurse) Zippers. Resolve IDs / URNs Mappings.
* Relation Statements: Match Resources / Reified Roles Templates.

## Augmentations Layers

* Interfaces (Sets / CSPOs Roles). Kinds aggregate Resources, Resources aggregate Occurrences, Occurrences aggregate Kinds.
* IContext : measurement contexts. Statement (data / state), Mapping (schema), Transform (behavior) contexts.
* Reification: members of Kinds / Occurrences implements super sets types. Kinds of type implements that type. ToDo: resource or occurrence interfaces in statements signatures. Class patterns (multiple interfaces).
* Abstract interfaces: ISubject, etc. Align interfaces to CSPO roles (traversal / graph layout)
* Order:
* StatementKind: PK(SK, OK). Relationship(Roles). Context, State, Mapping, Transform Kinds.
* SK(PK, OK)?
* OK(PK, SK)?
* Statement: abstract assertions (parsed / inferred). Mapping: abstract schema. Transform: abstract behavior. Relationships (dimensional / discrete): core model / ontology, Statement, Mapping, Transform synchronized (input / inferred Statement Events are fully parsed from CSPO Sets Layer).
* Order. States (Statements), Flows (Mappings), Events (Transforms). Kinds hierarchy tree / lattice (FCA). Action / Passion / State order. Kinds / Mappings domain / range Aggregation, Activation, Alignment. Comparisons. DCI / MVC / Relationships / Dimensional Aggregated upper onto matching gestures / flows.
* Inputs / Canonical: Match Interfaces / Signatures: (Context, Occurrence, Attribute, Value); Attribute / Value Roles in matching interface context. Order / hierarchy encoding / functions (sorted wrapped functional collections: wrappers set comparators / aggregation axis).
* DCI Context Model:
* DCI Relationship Contexts: Data / Schema / Behavior Model. DCI / MVC / Relationships Upper onto matching: gestures / flows.
* Metaclasses: PredicateKind SubjectKinds / ObjectKinds.
* Relationship: (Relationship, SubjectKind, PredicateKind, ObjectKind);
* PredicateKind of SK / OK. Employment(Employer, Employee); Employment (Employee, Position);
* Relation : (Relationship, Statements / Context, Role, Occurrence);
* Role : (Relation, Resource, Occurrence, Metaclass : Kinds);
* Occurrence : (Role, Relation, Context / Relation Statements, Resource);
* Discrete N-ary Relationship Aggregated Statements:
* Aggregated Statements traversal: expanded SPO form.
* Context: (Relationship : Predicate Kind, Relation : Statements, Role : Kind, Player : Resource);
* Predicate Kind of Reified S SK, O OK. (Relationship: Employment, Roles: Employee SK, Employer OK). Employment PK aggregated by Subjects and Objects Kinds. Relation Statements: Aggregated SK, PK, OK by Contexts Statement Kinds.
* (Working, workingRelationStmt, employer, IBM);
* (Working, workingRelationStmt, employee, John);
* Dimensional Relationships:
* (Dimension : Relationship, Measure : Relation, Unit : Kind, Value : Resource);
* (Time, oneHourStmt, minutes, 60);
* Distance Dimension: PK of Time SK / Meters OK. Define Dimension in terms of Relationship Kinds.
* Dimension Measure Statements: Domain PK Statements. Kind interface for Functional Transforms.
* Unit: PK Measure SK / OK Statement Kinds (SK / OK Members).
* Value: Dimension Measure Statement Kind Resource.
* Dimensional Order / Comparison: OrderKinds. Templates (populate).
* Order: Comparisons.
* Unit Measure Alignments:
* (Distance, travel, minutes, 60);
* (Distance, travel, hours, 1);
* (Distance, distance, km, 1);
* (Distance, distance, meters, 1000);
* Relationships alignments: To Do (PK hierarchies).
* Augmentations: Activation (Schema), Alignment (Data), Aggregation (Behavior) Matching (Mapping Function) results: Template Transforms (noop, merge, add); Transforms Flow State: listening for Matching Inputs.
* CSPO Inputs: Wrapper / Wrapped Core Statement Type / Instance Inference (data / schema / behavior):
* Parse into Core Model Templates (Raw Mappings / Transforms to be Augmented / Matched / Merged):
* Inputs Occurrences / Occurring Augmentations: Alignment, Aggregation, Activation of raw Template Inputs. Matchings / Merge (data, schema, behaviors)
* Inputs:
* Augment Resource with Kind in Context. Core Model Transforms Mappings Instances / Roles: Kinds and Singleton (Resource) Class.
* (Class : Transform, Instance : Kind T, Atribute : Mapping, Value : Kind U);
* Inputs Normal Forms: Dimensional, Discrete, etc. Parse Aggregations into Core Model.

### Aggregation Sail

* Type Inference: Kinds (Classes):
* Aggregate same Attributes occurrences for sets of Resources sharing same Attributes. Activate Context Transforms Kinds. Activate Kinds Resources Statements.
* Aggregations: ResourceURNs Source IRIs Sets / Layers streams / events (Resources, Occurrences, Kinds) parse / Occurrences population.
* Aggregation:
* Matching / Templates.
* Aggregation (Layers / Sets) Augment / Encode into Core Model:
* Inputs Normal Forms: Dimensional, Discrete, etc. Parse Aggregations into Core Model.
* (Class : Transform, Instance : Kind T, Atribute : Mapping, Value : Kind U);
* (Time, 1h, mins, 60m);
* (Working, 1h, USD, 40);
* (Working, 160h, USD, ?);
* (Employment, anEmployment, employee, John);
* (Employment, employee, Employee);
* (Employment, anEmployment, employer, ABC Inc);
* (Employment, employer, Employer);
* (John: Transform / Singleton, John, employment, anEmployment);
* (Employee, employment, Employment);
* (ABC Inc: Transform / Singleton, ABC Inc, employment, anEmployment);
* (Employer, employment, Employee);
* (John, employmentAt, ABC Inc.);
* (Employee, employmentAt, Employer);
* (ABC Inc, employsFor, John);
* (Employer, employsFor, Employee);
* Augmentations:
* Data Match Model (Sets).
* Infer Kinds / Flows.
* Models:
* (Statement, Kind, Attribute, Value);
* (Template, Statement, Kind, Attribute);
* (Template, Context, Statement, Resource);

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.

### Alignment Sail

* Alignments: Aggregation traversal: ResourceURN URN IDs Model population. Merge / Matching, order / relations / contexts. Encoding (methods).
* Alignment:
* Matching / Templates.
* Assert: (Class / Transform, Resource : Kind T, Attribute / Mapping, Resource : Kind U);
* Query: (Class / Mapping, Resource, Attribute / Transform, Resource) : CSPOs;
* Core Model:
* Occurrence / Mapping Declaration: (Mapping / Class / Metaclass,  Resource / Instance : T, Transform / Occurrence / Context / Statement / Class / Metaclass, Resource / Instance / Role : U);
* Occurrence Object Member of Subject as Transform / Function Role. Instance : T has Member Instance : U.
* Occurring / Transform Application: (Transform / Class / Metaclass, Resource / Instance : Kind T, Mapping / Occurring / Context / Statement / Class / Metaclass, Resource / Instance / Role : Kind U);
* Occurring Subject Member of Object as Mapping / Function Role. Instance : T is Member of Instance : U.
* Schema Match Model (Graph).
* Infer Types / Hierarchies / Shapes.
* (Statement, Resource, Kind, Attribute);
* (Template, Statement, Resource, Kind);
* (Template, Context, Statement, Resource);

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.

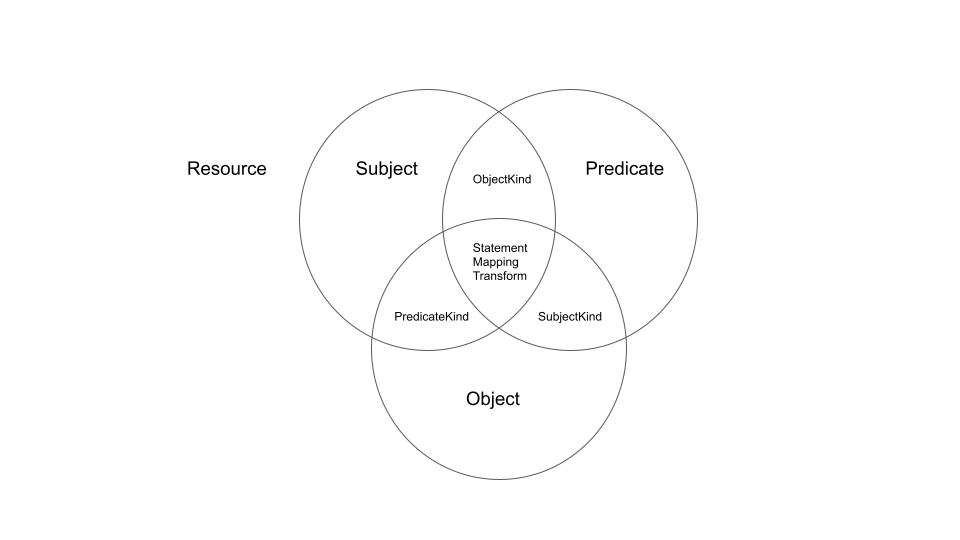
### Activation Sail

* Activation: type inference. Classification: determine metaclass / class / instance / occurrence subject roles for corresponding entity attributes and values. See Encoding.
* Activations: Relationship Models I/O (DCI Layers / expanded SPO Aggregations feedback). DIDs URN hashing / generation (HATEOAS Endpoints). Data Flow.
* Roles / Behavior Match Model. (Roles). Infer Contexts / Interactions / Measures.
* (Measure, Dimension, Unit, Value);
* (Resource, Measure, Dimension, Unit);
* (Statement, Resource, Resource, Unit);
* (Template, Context, Statement, Resource);
* (Roles with original wrapped Values).

## Templates

* Forms / Flows Dialogs / Contexts. Protocol. Resources, addressing, representations, navigation / traversal: properties "graph" rels (Wrappers / Transforms). Functional APIs.
* 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.
* Shapes. Relationships: Discrete (Organization, Situation), Dimensional (Measures arrangements). DCI / ISO W3C n-ary encoding. Expanded statements encoding. Apply Template, Template match: dataflow (roles, variables, wildcards, placeholders). Functional dataflow: order, comparisons (axes), workflows (state flow: referrer, URN, rel, target encoding). Revisions: graph versioning (E-Tag).
* DDD DOM HATEOAS Browsing Session / History object. Facets, key / value annotations. Current state available previous / next data / schema / behavior Gestures: DOM Functional Template flows.
* Integration: Distributed Consistency. Single distributed 'state'. Client / server, P2P actor / role / data (contexts: exchanges, request / response addressable interactions) validation. State 'exchanges'.
* Shapes: activation, dependency resolution. Class: possible number of instances (definition by extension / intension: digits / encoding).
* Ternary (balanced / cyclic) encoding (tensors) numbering system. Grammars / embeddings: digits / numbers (sign / concept / object). Event sourcing / blockchain / distributed consolidation (alignments) without distributed / centralized ledgers (seeds / hashs in state exchanges).
* Components (OSGi, Vert.x, JMX / JAF: signatures, activation, distributed resolution).
* Threads IO (streams, synchronization, ‘reference’ events).
* Augment alignements with 'feeds' (Solid, LOD, etc) connectors.
* Annotations (classes / individual subjects): attributes / values in occurrence contexts (facets / classes by attrs val aggregation).
* Contexts: occurrences in roles.
* Templates: reify Transforms. Aggregate Template Statements: mappings Resource value domains, predicates, ranges. Matching / Dataflow.
* Template Statement: Statements aggregates Kinds Occurrences for Context Resources.
* Template : Statement : Resource.
* Resource: Matching Mapping Statement Contexts. Resource Roles Mappings. Resolve Resource Role Monad for each Transform Quad DTO CSPO. Matching: Resource declarative Mappings aggregation.
* Statement: Apply Matching Contexts Mappings. Transform input Statements into CSPO / Graphs Role Alignments Mappings.
* Template: Reified Mappings. Aggregate matching input Statements. Graph / Models / Augmentations declarative (Mappings) Transform statements.
* Template Match: Patterns. Aggregation. Reified Roles Templates populates source Statements corresponding Role values mappings. Multiple Patterns Template aggregated as Statements Resources.
* (Template, SK, PK, OK) : Template;
* Template: Context Resource Monad expression (reified Roles). Aggregate Mappings.
* Multiple (aggregated) Template Statements renders different Statements to be merged for different Resources.
* Statement Apply: Render Template Pattern Statements given matching input Statements.
* Resource Resolution: Template / Statement matching. Aggregated Templates merge.
* Aggregated Template Statements. Populate from upper ontology inferred from inputs / Mappings invocation.
* (SK, PK): Possible Mappings.
* (PK, Empty Object): New Mapping.
* Statement : Monad(Statement);
* Statement : Resource.
* Statement: CSPO / Templates Mappings Transforms.
* Statement: Monad of Resource Quad (DTO).
* Statement Values : Context aggregated CSPOs.
* Statement as Resource:
* (C, (CSPO), P, O);
* (Amar, (Empleo, Pedro, Role, Manager), Ama, María);
* Matching / Aggregation: reified statements, kinds. roles, variables, wildcards.
* Matching: Patterns. Template Kinds matches Statements Kinds. Activation. Reified Statements Shapes.
* Matching Alignment. Apply Expansion: CSPO Transforms Statements results.
* Resources: Statements, Model Layers CSPO Roles. Resource Monad types / values. Uniform transforms API between models roles augmentations browse. (SAILs).
* Models:
* Sets: (Statement, Kind, Attribute, Value);
* Graph: (Context / Type, Resource, Attribute, Value).
* Graph: (Context : Type, Subject : Node, Attribute : Arc, Value : Node);
* Graph: (Resource, Context / Type, Attribute, Value).
* Roles: (Class, Instance, Occurrence, Role);
* Roles: (Metaclass, Role, Context, Instance);
* Discrete Roles: (Relationship : Metaclass, Relation : Role, Kind : Context, Resource : Instance);
* Dimensional Roles: (Dimension : Class, Measure : Instance, Kind : Occurrence, Value : Role);
* MVC / DCI: (Resource, Data, Context : Form, Interaction : Behavior); Layer Templates. Augmentations.
* Augmentations:
* Template Statements: Matching (Data), Aggregation (Layers), Expansion (Alignment).
* Matching: matching reified role classes hierarchy. Reified Relations: common superclass, order, others. Comparisons / roles.
* Data Activation:
* Augmentation: Sets Activation. Populate Models and their aggregated SPO inputs Statements. Aggregate SPO Kinds: type inference and matching. Render (Type, Resource, Attribute, Value) Statements. Type: Reified Kind (SPO), Attribute / Value of complementary SPO of Resource. Render (Resource, Type, Statement) Statements. Statement: Reified Statement in corresponding set. Data Alignment. Sets / Graph Models.
* Schema: Layers Aggregation. Layers Templates matches previous Layer Value with current Layer Context.
* Augmentation: Layers Aggregation. Populate Layers and their aggregated schema values context types. Different Models Aggregations yields equivalent Layers Statements shapes arrangements of Aggregations Models Schema roles. Layers Aggregation. Graph / Roles Models.
* Behavior: Layers Alignment.
* Augmentation: Behavior: Alignment. Entail aggregation Layers Statements expanded CSPOs. Entail Data Flow. Functional Data Flow Contexts. Expanded (possible / matchings) Statements traversal. Behavior Alignment. Discrete / Dimensional Models.
* Layers: Models Aggregation. Layer Contexts: upper previous layer Resource value wrapped in Context Resource Monad.
* (Amantes, Pedro, Ama, María);
* (MariaAmadaPor, Amantes, Pedro, Ama);
* (Amar, MariaAmadaPor, Amantes, Pedro);
* (PedroAmaA, Amar, MariaAmadaPor, Amantes);
* (Amor, PedroAmaA, Amar, MariaAmadaPor);
* Inputs: Aggregate SPO into CSPO: Aggregates Contexts Type / Table / Class Kinds. Aggregate PK Cols, Cols : Occurrence, Val : Resources.
* Inputs (Rel / Graph): (Type / Table / Class, PK : Resource, Col : Occurrence, Val : Resource).
* Inputs (Rel / Graph) FKs: Val : Resource equivalent PKs.
* Augmentations / Transforms:
* Data matching. Resource equivalence: identity / comparisons / order transforms.
* Schema matching. Predicates equivalence. Domain / Range types. Order: data flow contexts.
* Behavior matching. Domain / Range values applied functional predicates identity. Order: data flow interactions.
* Transforms: Resource Roles browsing, i.e.: Resource Template context Roles getters.
* Data layer: matching alignment transforms: Resource / Services augmentations.
* Template Aggregation Statements Types:,
* (Kinds / Types, Object / Resource, Predicate / Arc, Object / Resource);
* (Object / Resource, Kinds / Types, Predicate / Arc, Object / Resource);
* Encoding / Matching:
* Functional Primitives: (Matching). Graph Shapes Model. Layers. Example: reify / render / match "uncle" relation / "marriage" situation from graph statements.
* Primitive slots template Quads (Roles) populating "shapes" matching promps / flows / state transitions.
* Templates: Relation Statements Resources. Reified Meta Model.
* Order: Functional Dataflows. Functor / Transform domain / range Activations.
* Relations: Reified Monads / Transforms Mappings (data, schema, behavior) levels. Wrapper Type, Wrapped Type, Wrapped Value, Transform domain / range. To Do.
* Metaclass
* Class
* Instance
* Context
* Role
* Occurrence
* Primitive Relations.
* Functional Transforms: Browse State / Traversal. Navigational Context (Primitives State) Builder of Mappings (Templates).
* Augmentations: Matchings.
* Abstract Functional Quads encoding / representations.
* Functional DOM REST HATEOAS Facades. Discovery.
* Upper Ontology: Need, Product, Good, Purpose
* Upper Ontologies: From Primitives to Forms / UI Gestures.
* Units of Measurement (continuos) APIs /  Ontology
* Discrete (events) APIs / Ontology
* Templates / Transforms:
* XML / XSLT Like for Semantically Aggregated Layers of Statements:
* Reify Layers (Data, Schema, Behavior) Positional Meta Model Roles (Metaclass, Class / Instance, Context / Role, Occurrence) as Predicates / Attributes. Resource Values. Templates Matching.
* Reify Statements, Kinds, Resources. Templates Matching.
* Reify CSPO. Reify Context, Node, Concept, Node. (Statement Types).
* Canonical Statements Reification / Statements Aggregation of Type / Subject Statements Types.
* Layers:
* Contexts
* Kinds
* Resources.
* Relation Statements: Into Augmentation: Layer wise Data, Schema, Behavior "Templates" roles.
* Relation Statements: Match Resources / Reified Roles. Reified Meta Model Roles Resources. Resource "patterns", resolvable / reactive.
* Reify Relations / Predicates. Relation Mappings. Meta Model Positional Roles / Resources Matching.
* Templates / Transforms:
* XML / XSLT Like for Semantically Aggregated Layers of Statements.
* Layers Augmentation. Data, Schema, Behavior Layer wise Statement Roles.
* Alignment: Data Matching Augmentation.
* Activation: Schema Matching Augmentation.
* Aggregation: Behavior Matching Augmentation.
* Transform:
* Data: Model.
* Schema: Match. View / Context. Multiple ordered / recursive matches of Data Templates.
* Behavior: Apply. Controller / Interaction.

## Augmentation Layers Object Model



* Interfaces (Sets):
* Resources:
* ISubject : IResource
* IPredicate : IResource
* IObject : IResource
* Occurrences:
* IContext : ISubject, IPredicate, IObject, ISubjectKind, IObjectKind, IPredicateKind
* Kinds:
* ISubjectKind : IKind, IPredicate,  IObject
* IPredicateKind : IKind,  ISubject,  Object
* IObjectKind : IKind, IPredicate, ISubject
* IContextKind : IKind, ISubject, IPredicate, IObject
* Resource<Sets>
* SubjectResource : Subject<Resource>, Resource<Subject> : Resource
* Input (SubjectKind): (SK, C, P, O)
* Output (SubjectResource): (S, C, PK, OK)
* PredicateResource : Predicate<Resource>, Resource<Predicate>
* Input (PredicateKind): (PK, C, S, O)
* Output (PredicateResource): (P, C, SK, OK)
* ObjectResource : Object<Resource>, Resource<Object>
* Input (ObjectKind): (OK, C, P, S)
* Output (ObjectResource): (O, C, PK, SK)
* Kind<Sets> : Resource
* SubjectKind : Kind<Subject>, Subject<Kind>
* Input (SubjectContext): (C, SK, PK, OK) / Composite SK(PK, OK) Statement
* Output (SubjectKind): (SK, C, P, O)
* PredicateKind : Kind<Predicate>, Predicate<Kind>
* Input (PredicateContext): (C, PK, SK, OK) / Composite PK(SK, OK) Mapping
* Output (PredicateKind): (PK, C, S, O)
* ObjectKind : Kind<Object> , Object<Kind>
* Input (ObjectContext): (C, OK, PK, SK) / Composite OK(PK, SK): Behavior
* Output (ObjectKind): (OK, C, P, S)
* Context<Sets> : Kind
* SubjectContext : Context<Subject>, Subject<Context>
* Input: (C, S, P, O)
* Output (SubjectContext): (C, SK, PK, OK) / Composite SK(PK, OK) Statement
* PredicateContext : Context<Predicate>, Predicate<Context>
* Input: (C P, S, O)
* Output (PredicateContext): (C, PK, SK, OK) / Composite PK(SK, OK) Mapping
* ObjectContext : Context<Object>, Object<Context>
* Input: (C, O, P, S)
* Output (ObjectContext): (C, OK, PK, SK) / Composite OK(PK, SK): Behavior.
* CSPO Inputs. Hierarchy (classes) populate aggregations upwards from CSPO Contexts. Layer produced statements from aggregation of previous layer productions.
* Kind<Kind<Subject<Context>>> : StatementKind: Resource. Kind of Kind: SK(PK, OK). Contexts Kinds
* Contexts: Resources, Kinds, Occurrences: Statement (relation data), Mapping (schema), Transform (behavior) Contexts: composite Kinds: SK(PK, OK), PK(SK, OK), OK(PK, SK) respectively.
* SubjectContext: Statement. Data. SK(PK, OK).
* PredicateContext: Schema. PK(SK, OK).
* ObjectContext: Behavior. OK(PK, SK).
* Functional Sets Relations. Contexts: Statements (Data) / Mappings (Schema) / Transforms (Behavior)..

Sets (Quads):

Sets (Resources, Subjects, Predicates, Objects, SubjectKinds, PredicateKinds, ObjectKinds, Statements: Mappings / Transforms) abstraction for representing Augmented RDF Graphs.

Domain Model Object Hierarchy:

ClassName :: (aggregatingClass, subject / instance, attribute / predicate, value / object);

SPO/Kinds Set: Contexts (metaclass, class, instance, context, occurrence, role, etc.)

OntResource model Quads hierarchy:

OntResource: Universe Set.

(OntResource, OntResource, OntResource, OntResource);

Subjects : OntResource

(SubjectKind, Subject, Predicate, Object);

Predicates : OntResource

(PredicateKind, Subject, Predicate, Object);

Objects : OntResource

(ObjectKind, Subject, Predicate, Object);

SubjectKind (SK) : Subject. Predicate / Object Intersection.

(Statement / ParentKind, SubjectKind, Predicate, Object);

PredicateKind (PK) : Predicate. Subject / Object intersection:

(Statement / ParentKind, Subject, PredicateKind, Object);

ObjectKind (OK) : Object. Predicate / Subject intersection. Occurring.

(Statement / ParentKind, Predicate, Subject, ObjectKind);

Statements : Kinds / SPOs

(Kind, Resource, Resource, Resource);

Template : Kinds / SPOs

(Mapping, Kind, Kind, Kind);

Mappings : Kinds / SPOs.

(Transform, Context, Role, Occurrence);

Transform : Kinds / SPOs

(Context, Statement, Kind, Resource);

Services Facade:

MVC DCI REST HATEOAS / Functional APIs. Merge into OntResource APIs.

Class: Relationship (PredicateKind, SubjectKind, PredicateKind, ObjectKind);

Metaclasses: PredicateKind SubjectKind / ObjectKind.

Context : (Relationship, Statements, Role, Occurrence);

Role : (Context, Occurrence, Metaclass, Resource);

Occurrence : (Role, Context, Relation, Instance : Resource);

Aggregated Statements:

Context: (Relationship : Predicate Kind, Relation : Statements, Role : Kind, Player : Resource);

Predicate Kind of Reified S SK, O OK. (Relationship: Employment, Roles: Employee SK, Employer OK). Employment PK aggregated by Subjects and Objects Kinds. Relation Statements: Aggregated SK, PK, OK by Contexts Statement Kinds.

(Working, workingRelationStmt, employer, IBM);

(Working, workingRelationStmt, employee, John);

Relationship Relation Statements: Domain PK Statements. Kind interface for Functional Transforms.

Relationship: sameAs Statements.

Augmentations:

Contexts matching Statements applied to aggregated Mapping Context Transforms.

Apply Mappings Transforms. Transform Values Statement (Transform interface reifies Value as Statement Resource).

Order: Kind interface. Kind / ParentKind hierarchical order relation: more abstract / more specific hierarchy tree nodes until singleton Kinds (Order Statements, Templates, Mappings, Transforms as occurrences of Kinds / ParentKinds). Same hierarchy level ordered by ParentKind : previous, Kind : next relation.

Relationship Order / Comparison. Kind interface for Functional Transforms / Mappings (axis): parent, children, greaterThan, equals, lesserThan (compose nextSibling / prevSibling) via Functional invocation composition.

Dimensional Relationships:

(Dimension : Relationship, Measure : Relation, Unit : Kind, Value : Resource);

(Time, oneHourStmt, minutes, 60);

Distance Dimension: PK of Time SK / Meters OK. Define Dimension in terms of Relationship Kinds.

Dimension Measure Statements: Domain PK Statements. Kind interface for Functional Transforms.

Unit: PK Measure SK / OK Statement Kinds (SK / OK Members).

Value: Dimension Measure Statement Kind Resource.

Dimensional Order / Comparison. Kind interface for Functional Transforms / Mappings (axis): parent, children, greaterThan, equals, lesserThan (compose nextSibling / prevSibling) via Functional invocation composition.

Implement Functional APIs:

Activation (Data)

Aggregation (Schema)

Alignment (Behavior)

Implement recursion, aggregation, order, data flow, activation, alignment.

Domain Type Hierarchy: Reification, Resource Functor Transforms Domains: subtypes transforms wrapped compatible with results wrapped types by inheritance.

Type Inference: Kinds (Classes):

Aggregate same Attributes occurrences for sets of Resources sharing same Attributes. Activate Context Transforms Kinds. Activate Kinds Resources Statements.

Wrapped Types (Kinds) Inputs Inference / Matching. Wrappers contains Wrapped CSPO Role Resources. Functional Flow into Occurrences, Attributes, Values.

Encodings. Representations: Instances / Literals Encoding. URNs. Resolution: sameAs Mappings / Parsing. Occurrence / Occurring domainOf / rangeOf Type Inference.

Model Kinds: Model Reified.

Domains Kinds: From inputs.

Reified Model Resource Kinds.

Functional: Monads (wrappers types / wrapped types inference). Kinds Domain Flow (Mappings):

DOM Resources: dynamic object model / kinds.

Model API:

Inputs / API:

I/O Normal Form: Statement

Service Facade. Functional Data Flow: Matching Mapping Transform: Statements. REST HATEOAS URNs:

I/O Statement:

(Context / Class, Instance, Attribute, Value);

Data Flow: Service Facade API:

REST Data Flow: Services Facade URN request / response HATEOAS flow.

Transform::Mapping::Statement::Kind::Resource;

Resource::Kind::Statement::Mapping::Transform;

Sets Resources REST HATEOAS / Data Flow IO Model Statements:

(Transform, Mapping, Statement, Kind);

Functional Data Flow:

Transform::Mapping::Statement::Kind::Resource;

Resource::Kind::Statement::Mapping::Transform;

Encoding.

Augmentations:

RDF Backend. Event sourcing (bus) saga pattern. Publish / Subscribe. Connectors.

Data Matching: Activation.

Schema Matching: Aggregation.

Behavior Matching: Alignment.

Activation (Data Matching):

RDF Quads Parsing from events sourcing events bus:

(Class, Instance, Attribute, Value);

Populate SPOs / Statements / Kinds / Mappings / Transforms Quads Wrappers Sets Objects for Aggregation.

Ontology Matching: Resources Kinds Matching. Merge same URNs.

Aggregation (Schema Matching):

Aggregation. Quads CSPOs / Attributes / Values. Handle recursion. Functional Transforms Context: subjectKind::subject::subjectKind (same subjectKind).

Schema Matching: Aggregation Kinds Matching.

Resources aggregate into Kinds. Kinds aggregate into Statements, Statements aggregate into Mappings. Mappings aggregate into Transforms. Hierarchy aligns Wrapper types reification.

Quad Wrappers (Resource hierarchy) wraps aggregated occurrence of wrapped Quad Type. Wrapped Quad Type: Kind. Wrapped: DOM / DTO of Kind members.

Alignment (Behavior Matching):

Resources Reification: Kinds, Statements, Mappings, Transforms reified. Reified Resources aggregates aligned into Transform Wrapped Quads:

(Kind, Statement, Mappings, Transform);

HATEOAS Functional Browsing. RDF Model Serialization.

Behavior Matching: Transform Quad Kinds Matching.

* Notes:
* URN : Resource (alignments). Primitives.
* Resource : Root Category. URN : Source / Surrogate Key / Crafted. Naming / Encodings (below).
* Ontology alignments: Data / Schema / Behavior Augmentations. Model / Schema / Upper / Domains: purposes / gestures (MVC / DCI Mappings / Transforms) layers. Example:
* Occurring / Context (Statements / Kinds)
* Roles (Metaclass, Class, Occurrence, Context, Role)
* MVC / DCI Mappings / Transforms. Example: Forms, Purpose, Gestures, Actors, Roles. Data / Schema / Behavior alignment.
* ESB: Endpoints, Features, Interfaces, Service Process Description / Discovery. Reactive Events Subscriptions. HATEOAS Endpoints "autowiring".
* BPM: Process, Steps, Flows, etc.
* Augmented Actionable (Process Flows, Items Activation) CMS. Browser: HATEOAS Protocol / APIs / Augmentations. Inferred / Reified / Resolvable Data Flows. Designer: Model Pallete. Declarative core / domains types / instances browsing / discovery "wiring".
* Graph Reified Grammars (upper). Contexts / Mappings. Terminal / Non Terminal. Rules / Productions. Mappings / Transform: browse grammar, rules, productions:
* (Rule, Context, lhs, rhs)
* Naming: Kinds / URNs Addressable Encodings. Parsing: URNs Encoded Functional Distributed Resource Resolution. Data Flow Transform / Mappings: Embedded Productions: Augmentations. NLP / NER. Ontology Matching: URN Class Transforms.
* Graph Embeddings: ML Backend Services (ML Predictions Augments Mappings / Transforms). Encodings (Naming).
* Encoding: Deep ML Embeddings. Data: classification, Schema: clustering, Behavior: regression.
* Naming: Auto Encoders. Semantic Hashing. Resources Mappings / Transforms Reified Maps / Tables. Keys / Values Resource Hashing / Resolution Functions: Contextual to Functional Environment State: Mappings Flows / Wrapped State.
* Naming: Augmentations. Contextual Hash Enabled: Functional Mapping Flows Map / Table Encoded / Resolved. Functional Relations: Ontology Matching / Aggregation / Inferences by Hash Encoded Metadata / Transforms Resolutions.
* Clients / Browsers: Peers. Protocol: Reactive Dialogs Prompts. Events. Distributed Data, Schema, Behavior Core Model Statements Encoded I/O: Layers Sync / Augmentation of Knowledge requested from each Peer(s) as Model inputs given resolution of Dialog (Subscriptions) event sourcing state. MVC / DCI Distributed State Transforms / Mappings. Augmented Peer(s) Models: updated View State (flows) / Mappings / Transforms. Rendezvous Peer Role. Local Peer: APIs for local / remote views (MVC / DCI) views (Web, REST) Rendering.
* Supertype / subtype: kinds, contexts, statements
* Context, statement, mapping, transforms Data flows. Order relations / mappings.
* (...)
* Integration / Alignments: OntResource I/O Adapters. Smart ESB (Subscriptions / Dataflow). Augmentations.
* Services Facade: OntResource: gettets metaclass, class, instance, context, occurrence, role in context. MVC DCI: HATEOAS Functional Domain. Generic REST Object Viewer / Browser. Activation. Declarative Services Endpoints (saved queries / state flows). Data Flow Forms: Transforms specs. Order / Facets.

# Services

## Services Layer

### Registry Service Sail

* Registry Service:
* ResourceURNs Statements bindings:
* Registry::resolve(ResourceURN) : dispatch to matching signatures:
* Registry::resolve(ResourceURN) : Resource
* Registry::resolve(ResourceURN) : Kind
* Registry::resolve(ResourceURN) : Occurrence
* Registry::resolveResourceURN(Resource, Occurrence, Kind) : Resource ResourceURN
* Registry::resolveResourceURN(Kind, Occurrence, Resource) : Kind ResourceURN
* Registry::resolveResourceURN(Occurrence, Kind, Resource) : Occurrence ResourceURN
* Reactive abstractions over P2P layer. Producer, Consumer, Processor, Message. P2P Browser. Platform. Container (devices layer).
* Deployment. Interfaces (endpoints: client / server, APIs for reactive abstractions). Network: interfaces, addressing, messages, protocols, routes implementing services / augmentations (encodings).
* Distributed CAM. Encoded in devices. Event sourcing (sync: indirect knowledge predictions / inferences). Augmentation: assertions / rules.
* Interfaces / QA / Wizards / Queries: which day of which month was the Wednesday before of last year Easter and how many patients where diagnosed of flu that day and whose of them where smokers and whose of them lived near some region being workers of some factory.
* P2P Browser: search session history. Referrers. Purpose steps. Role navigation bookmarks. User interface (declaratively domain driven protocol / queries / assertions).
* Content type activation: smart semantic overlay endpoints, metadata, description, addressing, linking, augmentation. Application and filesystem augmentation. Syndicated file, application, content, behavior manager. P2P Browser. Universal Plugins (Protocols, APIs, implementations): Assisted Contexts P2P Browser. Inter domains oriented.
* WebDAV virtual filesystem plus index, registry and naming description / linking / addressing metadata services on device for P2P Browser storage / retrieval / activation. Ont DIDs services. DID Endpoints: WebDAV, OData, HAL, SPARQL, other protocols (WS, RPC, etc.).

### Naming Service Sail

Naming Service:

Input IRIs Encoding / Hashing. Input IRIs Matching. Endpoints (Messages Signatures). Semantic Hashing: DIDs. HATEOAS: Workflow states / referrers.

### Index Service Sail

Index Service / Logs:

Query Graphs of ResourceURN Nodes / Messages. Events driven Persistence.

## Services Layers Object Model

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).

Resources monadic pattern:

Class<Observable<OccurrencesSubClass>> extends SuperClass;

OccurrencesSubClass extends Class;

Observable: stream of Class occurrences in OccurrencesSubClass (as a CSPO member);

Class / OccurrencesSubClass CSPO members: (C: SuperClass, S: SuperClass, P: SuperClass, O: SuperClass);

CSPO members: SPO roles / kinds / context metadata (calculated / aggregated into kinds resources) accessible from convenience methods.

CSPO members occurrences resolution / aggregation by metadata / context.

Example: Statement is a subset of Resource occurring in a Kind. Resource is an URI occurring in an Statement.

Kinds (schema layer). Class, Metaclass (types) aggregation of attributes, occurrences, values. Resource: Statement URI attributes / values.

Monads API:

URI<Observable<Resource>> extends String;

Resource<Observable<Statement>> extends URI;

Statement<Observable<Kind>> extends Resource;

Kind<Observable<Flow>> extends Statement

Flow<Observable<Dimension>> extends Kind;

Dimension<Observable<Measure>> extends Flow;

Measure<Observable<Metamodel>> extends Dimension;

Metamodel<Observable<URI>> extends Measure;

Hierarchy: instance: previous statement, class: next statement, metaclass: occurrences. Class hierarchy: every class reified as instance of previous statement class.

Activation / aggregation on layers.

DCI Layers: dimensional, data, schema, behavior. CSPO roles.

CSPO members: (C: Class, S: OccurrenceSubClass, P: SuperClass, O: SuperClass metaclass);

Bound functions: aggregate / de-aggregate, functional activation, transforms, filters, predicates, kinds, typed (roles: dimensional data, schema, behavior / instance data / kind relations) traversal / joins, others.

Versioning / CRUD: context relations determines current / role entities occurrences.

* Modal logic. Sets. Categories. Groups. Formal spec (data / reference models).
* Data structure
* Instance / behavior
* Wrapper 'monadic' domain: dynamic wrapper type (set / groups).
* Set Membership / Set Operations 'dynamic' domain.
* Dynamic dataflow, operation concatenation, matching (signatures). Injection (references resolution / roles), arrangement in contexts (embeddings: role / rel between members arrangement, navigation, resolution, operations: dynamic monadic types). Dynamic DOM (dimensional DCI, data / info / knowledge 'contexts': signatures refs resolution; sign / concept / object contexts: resolve elements / operators).
* Wrapper types (Token):
* Resource: dynamic type wrapper of Token (occurrence of Token in Resource). Class, Metaclass, Roles. Tensor like internal representation.
* Resource (token, C, S, P, O Resources: mappings / tensor like CSPO dimensional arrays). One instance for each context occurring in subjects with attributes / values.
* Dimensional DCI (data, schema, behavior)
* Positional roles wrapper types (Resource interfaces):
* Context (Sign, Concept) : Object
* Sign (Context, Concept) : Object
* Concept (Context, Sign) : Object
* Object (Context, Sign) : Concept
* Statements (object):
* (Context, Sign, Concept, Object)
* Statement (object) layer (DCI) type: wrapper types in positional roles.
* Layers: dimensional, data, schema, behavior.
* Dataflow: objects (statements) mappings.
* Mappings (retrieve existing or create new: functional):
* Context<T, V>(Context<T, V>) : Context<T, V> (assert data: price, infer info: variation new 'column', perform knowledge: assign roles).
* Mappings: 'forms’ serialization code / data format (homoiconic LISP like). (TID:VID (TID:VID (TID:VID))) assertion / query. Parses 'monadically’ functionally into Resources manipulations (activations). RDF triple serialization (SPO), expression resolves into Context Resources.
* Dataflow: internal Resource representations using Tensor like datatypes.
* Dataflow: statement (forms) assertion fires mappings resolutions (align / augment: infer / resolve created / retrieved references old / new statements; infer sameAs / rels / ctxs: sets, synsets, embeddings, categories, groups).
* Dataflow: Factory / builder API (Tokens / Resources).
* Dataflow: Facade (reactive inputs / outputs streams).
* Dataflow: Facade (query, browse, index).
* API: Services.
* Frontend / API: upload CSV / XML / JSON (service / form).
* Frontend / API: ontology browser, (faceted) search, index, export formats (queries / sync). Dynamic forms generation (behavior / schema / data).
* Frontend / API: batch / streaming (services) IO adapters / mappers (functional / streams) from / to CSV, XML, SQL, JSON, SemWeb, etc. to internal mappings 'forms’ serialization format (XSLT, DOM / SAX event driven like). Integration: aggregation and alignment (store state for batch / services administrable contexts / namespaces).
* Declaratively state 'pages’ (applications / DCI / MVC contexts use cases / flows) by declarative statement (internal mapping 'forms') of dimensional classes / instances of behavior / schema / data metamodels via de-referenceable (evaluable) resource streams. Application metamodel data 'upper’ ontology plus domain data ontology: rendering (services / clients) via XML (schema) IO / XSL template libraries. Content type 'activation’ (available 'forms’ queries / assertion according resource state flows).
* Dataflow: 'forms’ queries / assertions: context, member and 'types’ that could resolve / fit into 'activation’ arguments (arrangements).
* Set classes layers (contexts: Behavior sets parents of occurrence layers):
* Dimension
* Unit
* Measure
* Instance
* Occurrence
* Kind (roles)
* Class
* Flow
* Behavior

### Meta Model

* Metamodel resources activation: aggregation, alignments, layers ontologies (backends functional wrappers: IO / custom logics). Resource activates on messages arrival (dataflow nodes / arcs signature / patterns / template routes / dispatch): graph activation (dataflow to corresponding 'listener' signatures). Response flows to declared dataflow / patterns 'listeners' (referrer / contexts) resources.
* Aggregation alignment.
* Identity alignment: Metaclass, class, instance, occurrence / template: dimensional (layers) identity alignment / entity blend / merge / data / schema / behavior match. Classification (dimensional state / layers). Data (reified).
* Attributes alignment: Clustering (dimensional state / layers). Schema (reified).
* Roles alignment: Regression (dimensional state / layers). Behavior (reified).
* Encoding: resource addresses as relative 'pointers' (template / occurrence pairs resolutions / rels routes, transform identifiers). Contexts: dimensional instances of types in roles. Composition of resource data / occurrences in resource templates (dataflow activations). Semiotics: contexts, signs, concepts, objects.
* Resource metamodel: quad roles (dimension / context, occurrence / behavior, attribute / schema, value / state / data). Aggregated into messages / metamodels (entity, schema, behavior merge).
* Statement roles (dimensional, data, schema, behavior) 'relative’ to contextual evaluations (functional, ordering, reification). Class, metaclass, instance context / occurrence templates: context hierarchy behaviors before dimensional (occurrence of) dimensional statements.
* Resource address: context address + occurrence address (addresses, recursive + attr / val address contexts).
* Resource representation: occurrence in context (addresses) attributes / values (addresses): ctx / occur attribute / value Statements (addresses).
* Address: quad (ctx:occur). Encoding.
* Address Resolution: (peter:x, emp:devel, :, :) : (peter:someDate, emp:softwareArchitect, salary:annual, 60000:USD); other attrs…
* Dimensional quads resource addresses context / occurrences / attrs / vals resolution: aggregated / inferred from context (browse state, link rels, referrer, etc.). Encoding.
* Activation: query contexts for kinds, flow occurrences, etc. resolution. Query for aggregation learning (listen possible dataflows).
* Type promotion. Role promotion (behavior, interface). Deterministic materialization: loops (reifications / occurrences roles). Definitions (templates). Reifications. Quads, triples, pairs (metaclass, class, instance, states: flows, behaviors).
* Activation wizards: assets (resolution, forms, flows). Dynamic assets 'prompts'. Dialog 'protocols' (slots three alignment kinds in / out nested contexts). Assets: ActiveRecord / queues. Chrome / XUL / SWT / ZK runtimes. Jupyter notebooks.
* Layers (metamodel): uniform modelling of domains and applications / services (parser). DCI / MVC 'gestures' (IO / activation / user, service interaction) as activation messages. Encoding, resource augmentation, lambda wrappers (bound functions in descriptions from metadata). Uniform layer interaction models. Events / queues (server push).
* Microkernel: workflow / scheduler. Input / output plugins, kernel plugins. Declarative blueprints / layers (CDI / IoC / domains discovery).
* CQRS. Event sourcing (rehydrate entity, descriptions somewhere).
* Microservices: from aggregated domains descriptions (augmentation, declarative template extensions, declarative plugins).
* NodeJS: meta-web description framework. Activation (descriptors pointers / lambda commands, runat contexts layers integrations). Serverless (codats).
* Streams. Reactive Programming. Dataflow. Endpoints (behavior) / content (schema) metamodel node / layer modelling. Signatures. Frameworks: messaging, networking, events, routing. Metamodel Activation Protocol. APIs protocols. Implementations. Deployment: containers / custom blueprints, management APIs.
* Domains. ERP / HMS / CRM / SCM / BI. Integrations (Activation).
* VM (RDF backend event driven, graph DB, TensorFlow shapes).
* Abstractions: Graphs, Edges / Statements, Nodes / Resources. Aggregations (Edge / Statement), Activations (Node / Resource). Edge / Node "expressions" (signatures) : declare aggregations / activations (Forms / Templates: possible Statement / Resource instance classes). Aggregation / Activations: Statement / Resource “occurrences”.
* Dataflow (declared routes / signatures: matching shapes / grammars) aggregations / activations.
* Aggregation: Statement(Resource) : Statement. Aggregates over Resource into Statement. Declare aggregation.
* Activation: Statement(Statement) : Statement. Declare activation.
* ID Alignment: Resource(Resource) : Resource.
* Attr Alignment: Resource(Statement) : Statement.
* Role Alignment: Resource(Statement) : Resource.
* Operations as Dataflow streams possible aggregations / activations. Translation / orderings / comparisons / aggregations.
* Models / graphs layers from Resource until Behavior, binded through semiotic Resource / Statement / Role IDs metagraph.
* Data (resources: syntax).
* Schema (dimensional: semantics).
* Behavior (application: pragmatics).
* Semiotic Resource / Statement / Role IDs metagraph.
* Naming, registry, index services APIs.
* Translation (domains). Subject, attribute, value, occurrence, role bindings.
* Primitive operations: sort in context, compare, aggregate, activate, alignments (ID, Attrs, Roles).
* Functionality: translate / integrate. Expose uniform distributed systems gateway. IO / facades (data, schema, behavior activation flows). Pluggable architecture (sources / endpoints: same aggregation, activation, alignment interfaces).
* Resource / Statement: Metamodel
* Template / Form: Declaration
* Activation / Aggregation : Behavior
* Data Metamodel
* Dimensional Metamodel
* Flow Metamodel
* Resource IDs Graph: Dataflow (Aggregations, Activations, Alignments).
* Executable Metamodels. Domains Translation. Unified Integration Dataflow Event Bus.
* VM / IO Connectors (client / service abstractions). Declarative Functional Data / Dimensional / Flow Metamodel Layers specifications (pub / sub streams) Templates / Forms. Activation / Aggregation "instances".
* Element: Observer / Observable (streams / filter / map / reduce).
* Node / Edge, Resource / Statement, Template / Form, Activation / Aggregation.
* Dataflow: matching element "templates" (grammars / signatures).
* Events: Element wise listeners (instance, class, occurrence). Produces new Elements (events).
* Events example: Forms produce Statements from Aggregations, Statements Aggregations and Aggregation Forms.
* Elements: publish / subscribe via addressing (observer / template signatures dataflow) to / from edges / nodes (Templates / Forms, Statements / Resources, Aggregations / Activations).
* Addressing: From Resource Quads Encoding. Resolve routes from / to via template signatures dataflow.
* Backends (support implementation of edge / node element level events):
* RDF Backend.
* Graph DB Backend.
* Reactive (Vert.x) Backend.
* TensorFlow Backend.
* Element API: Wrapper over backend implementation. Event handling. Base Metamodel hierarchy (grammars / schema: Dimensional / Role Kinds, data: layers, behavior / flows: Resource Graph IDs).
* Order (sort in context), comparison (class / instance ID): "common denominator" (jump / iterations). Activation /Aggregation. Read / Write Attributes.
* Data (Nodes / Semiotic), Information (Templates / Dimensional), Knowledge (Resources / Layers).
* Processor : Observer, Observable.
* Exchange : Subscription. Processor.
* Processors (Alignments):
* Nodes, Edges, Routes (hier.).
* Templates, Forms, Aggregations (hier.).
* Resource, Statement, Activations (hier.).
* Processors: instance of (type object) hierarchy. Activation, role kinds: Aggregation. Aggregation, paths: Routes.
* Declarative Aggregations: hierarchy type object pattern.
* Models: Processors. Collections (observables) of Activations, Aggregations, Routes.
* Triple store backend. RDF4J:
* Quad Layers SAIL. Core layers schema. IO: RDFS / OWL (classes, types, sameAs, etc.). SPARQL. Upper Ontology.
* Objects Layers SAIL. IO: Objects (Layers / DOM / OGM) HATEOAS.
* Functional Layers SAIL. IO: Augmentations: Browse / Navigate Model (Objects / Monads). Message Driven.
* Augmentations / Domain Services / Dataflow SAIL. IO: Messages built in Functional SAIL / Augmentation Contexts results streams. Contexts: Model Contexts, Domains Services REST Contexts URIs (APIs), connector / clients plugins (signatures). DIDs Domain Services: internal URIs. DIDs / URI mappings / APIs. Signatures. Bus.
* Augmentations: Perform Aggregation, Activation, Alignment according input Message. Update Model Contexts (upwards), Occurrences (downwards) according Augmentation stream results (Contexts). Enqueue further dataflow messages.
* Dataflow (Predicates, Mappings, Functions) domain / range (signatures). SAIL registry (types / kinds bus). Bus: topics, queues. Reactive interfaces. Dispatcher.
* Services SAIL I/O: Connectors / Clients. Distributed nodes Connector sources.
* URI: Jersey / CDI APIs. Persistence interface template methods.
* Layers APIs.
* Statement URI: Layer URI + Layer URI instance ID.
* Layers CRUD tests.
* Functional APIs.
* Functional tests (Inside Flat Map: dataflow over Object APIs):
* Predicate tests. Activation.
* Mapping tests. Aggregation.
* Function tests. Alignment.
* (MessageDomain, Predicate, Mapping, Function);
* Flat Map: Dataflow chaining over Message transforms and dataflow results.
* Augmentations performed over individual Message roles in dataflow and in dataflow with other Message roles (APIs / contexts).
* Message: Functional wrapper of Layers Contexts hierarchies statements. Hierarchy: template methods, predicate, mapping, function behaviors (inside flat map and on message statement role positions: internal dataflows).
* Predicate, Mapping, Function, URI, Statement, Value. Root hierarchy types.
* (Statement, OldEmployees : Predicate URI / Activation, SalaryUpdate : Mapping URI / Aggregation, Percentage : Function URI / Alignment
* Monad::of(URI instance : hier);
* Monad::flatMap(Statement) : Monad (of Statement hier).
* Dispatch: each layer instance consumes matching or forwards upwards (layers signatures / zippers) incoming messages. Resulting Message enqueued for further processing.
* URI::onMessage (template methods: context, occurrence, attribute, value, role, contexts, occurrences, etc.).

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.

Order: Common super type / kind / role / occurrences. SortedSet.

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).

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.

### Messages

* Context Resource produces occurrence / attribute / value events (aggregated by occurrence). Context URI events bus.
* Occurrence Resource consumes resource occurring events (in context, with attributes / values. Occurring URI events bus.
* Message: transform T > R (subclasses of URI: content types, APIs) available verbs / flows / navigation (state). Dynamic Resource monad over URIs.
* Message: Metamodel Resource / URI instances / API wrapped as reified Message. DOM. Object navigation.
* Activation Message: onSubject<Statement, Kind> materializes SubjectKind. Idem PredicateKind, ObjectKind.
* Aggregation Message: onLayer<Statement, Statement> materializes next layer Statement.
* Augmentation Message: ID / Classify, Attributes, Context Role.
* Contexts (dataflow): possible message flows invoked as available inputs / materialized results matches transforms signatures.
* Messages: dispatch. Processors. Event bus topics for each model layer types and for each resource type URI instance.
* Queries: layers traversal / aggregation (streams). Person as Employee in Sale of Offers.

### Dataflow

* Quads I/O: Base Model Protocol. URN Encoding / Content Types. Transforms / Mappings browse links transforms.
* Wildcards, variables, placeholders. Relationships intention / extension.
* Functional Facade stateful browse.
* Implementation: Runtime. Architecture. Components. Patterns. Models. Messages. Augmentation. Events dispatch. Message aggregation / instantiation / resolution / application. Backends. Services.
* Templates: producers / consumers shapes state / functional dataflow layouts.
* Resources Inputs / Outputs: Augmentation, Protocol, Browser. Message resolution / application.
* Component: Models (data). Source: augmented input statements. APIs (Model).
* Component: Messages (contexts). Source: augmented models templates. APIs (Model).
* Component: Transforms (interactions). Source: input statement case matching Message inputs. Returns / materialize results. APIs (Model).
* Core Model API: Augmented (Aligned, Activated, Aggregated inputs matching context messages) IO. Resource MetaGraph. Dimensional model. Grammar. Model repository. Backend. 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. API.
* Transform: Interaction Model API. Input statements: Message application results. Functional application of Message over Resource. Input statements: Resource over which applicate / augment with matching Message. API.
* 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. Materialize Transform. Results.
* Services: distributed addressing / resolution, reactive distrbuted event bus: streams / events, index, naming, registry.
* Monads. Wraps Models Roles. Matching: Patterns wrapped Resource Roles values: placeholders, variables, wildcards, reified instances.
* Core Roles (Wrappers):
* Resource : Monad(x : Resource);
* Statement : Monad(x : Statement) : Resource, Quad DTO.
* Context : Monad(x : Context) : Statement;
* Template : Monad(x : Template) : Context.
* Wrapper / Wrapped reactive / event driven message bus API.
* Quads I/O: Base Model Protocol. URN Encoding / Content Types. Transforms / Mappings: browse links transforms. Layers: Augments Models (resolve / merge ontologies).
* Augmentations:
* Data Match Model (Sets).
* Infer Kinds / Flows.
* Models:
* (Statement, Kind, Attribute, Value);
* (Template, Statement, Kind, Attribute);
* Schema Match Model (Graph).
* Infer Types / Hierarchies / Shapes.
* (Statement, Resource, Kind, Attribute);
* (Template, Statement, Resource, Kind);
* Roles / Behavior Match Model. (Roles). Infer Contexts / Interactions / Measures.
* (Measure, Dimension, Unit, Value);
* (Resource, Measure, Dimension, Unit);
* (Statement, Resource, Resource, Unit);
* (Template, Context, Statement, Resource);
* (Roles with original wrapped Values).
* Models:
* Augmentations:
* Sets. Data Match Model.
* Resource Context: Data Activation, Kinds Attribute Values matches layer Resources. Ontology matching.
* Functional Expansion Transform Mappings: Augmentations domain transforms, Resource Roles APIs.
* Match: (Template, Resource, Statement, Kind);
* Functional Expansion Transform Mappings: (Resource, Statement, Kind, Value);
* Graph. Schema Match Model.
* Statement Context: Schema Aggregation, Resource Kinds Attributes matches layer Statements.
* Match: (Template, Statement, Resource, Kind);
* Functional Expansion Transform Mappings (Statement, Resource, Occurrence : Kind, Attribute);
* Behavior: Reified Mappings / Transforms / Functions:
* Roles. / Dimensional / Discrete. Behavior Match Model. Match: (Template, Dimension, Measure, Unit : Kind); Aggregate Contexts and Interactions.
* Match: (Template, Dimension, Measure, Unit : Kind);
* Functional Expansion Transform Mappings: (Dimension, Measure, Unit, Value);
* Template Context: Behavior Alignment, Template Statement Resource Kind matches layer Template Roles.
* (Template, Statement, Resource, Kind);
* Layers. Apply Templates. Templates Matching populates augmentatiom layers: Data, Schema, Behavior matching Templates and applying Transforms: Models schema signatures (roles).
* API: Functional DOM Reactive / REST HATEOAS. Browse Monads Roles Mappings. Subscriptions: Materialized Template dataflow context bindings: Patterns.
* Resource Context: Data
* (Resource, Occurrence, Attribute, Value);
* Statement Context: Schema
* (Statement, Resource, Attribute, Value);
* Template Context: Behavior
* (Template, Statement, Resource, Attribute);
* Augmentations:
* Data Match Model (Sets).
* Infer Kinds / Flows.
* Models:
* (Statement, Kind, Attribute, Value);
* (Template, Statement, Kind, Attribute);
* Schema Match Model (Graph).
* Infer Types / Hierarchies / Shapes.
* (Statement, Resource, Kind, Attribute);
* (Template, Statement, Resource, Kind);
* Roles / Behavior Match Model. (Roles). Infer Contexts / Interactions / Measures.
* (Measure, Dimension, Unit, Value);
* (Resource, Measure, Dimension, Unit);
* (Statement, Resource, Resource, Unit);
* (Template, Context, Statement, Resource);
* (Roles with original wrapped Values).
* Aggregate Layers Contexts: Value Context Upper Shifting Layers. Model Layers Augmentations.
* Layer Aggregation: Perform Augmentation of Aggregated Layers Data, Schema, Behavior Templates. Functional Model Data Flows CRUD.
* Aggregation: Augment Aggregated Layers Data, Schema, Behavior Templates. Functional Model Data Flows CRUD.
* CDI: Signatures Functional Dataflow Injection / Bus Resolution.
* Spring / OSGi Blueprints
* ServiceMix / Fuse
* RxJava / JAX-RS
* Declarative settings: nodes, routes, patterns / transforms functional flows (monads): from Metamodel schema / instances data (URIs, Resources, etc.). Metamodel hierarchy layers (endpoints / routes / messages: URIs / Resources) as common Blueprint / Spring components (schema monads classes context / roles, interactions instance data).
* Messages: encoding / addressing. Signatures / patterns: dynamic discovery / flows. Bound functions: schema / domain DCI: declarative templates encoded in Metamodel (transforms).
* Graph encoding (Metamodel quads): Context, SPO, Kinds. Sets (recursive statement, kind, SPOs).
* Metamodel addressing: inputs / outputs of normalized URIs dataflow flowing through hierarchy from URIs till Metamodel till URIs (context driven): Aggregation, Alignments, Activation.
* Backends handles Metamodel URIs IO via Metamodel metadata / functional contexts.

### Functional DOM Model.

### To Do.

Services Layer Functional Dynamic Object Model: HAL, GraphQL, OData, endpoints protocols (Connectors). DOM wrapped values, dynamic functions domain / range (Kinds dataflow).

DOM DCI HATEOAS. Functional. Traversal: Prompts / Transforms: Dialogs.

Bus Signatures Dataflow Sail. Dynamic Monads. Zippers, Transforms from domains. Dynamic Services functional domain / range signatures / subscriptions matching.

DOM Resources of Resources: Occurrences / Occurring CSPOs Transforms Matching Wrapped / Wrappings / Transforms Resource Types Shapes Matching Templates. Data flow, apply transforms, order, lists.

Monads. Wraps Models Roles. Matching: Patterns wrapped Resource Roles values: placeholders, variables, wildcards, reified instances.

Reify Layers (Data, Schema, Behavior) Positional Meta Model Roles (Metaclass, Class / Instance, Context / Role, Occurrence) as Predicates / Attributes. Resource Values. Template Matching.

Reify Statements, Kinds, Resources. Templates Matching.

Reify CSPO. Reify Context, Node, Concept, Node. (Statement Types).

DDD (Meta Circular Interpreter) : Code as Data. DOM (Occurrence / Occuring):

Prompts / Flows: View Forms.

Meta Model : Dataflow

Protocol: codat I/O Use Cases

Statement:

(Context : Statement, Subject : Resource, Attribute : Kind, Value : Resource);

Value Resource according Kind (Roles) Statement Subject occurrence.

Sets: de aggregated Statements. Reification. Layers (Models) expands Statements for each Kind Attribute.

Monads. Wraps Models Roles. Matching: Patterns wrapped Resource Roles: placeholders, variables, wildcards reified instances.

Functional Data Flow. ResourceURN Events. Message Logs Streams / Traversal (Index Persistence Events Graph Interfaces):

(Occurrence, Kind, Resource)

(Occurrence, Resource, Kind)

(Kind, Occurrence, Resource)

(Kind, Resource, Occurrence)

(Resource, Occurrence, Kind)

(Resource, Kind, Occurrence)

# Topics

* Schedule. Tasks.
* Notes : Topics.
* Topics: P2P, Blockchain, DIDs. Ont DIDs.
* Topics: RDF(S), OWL, SPARQL, ShEx, SHACL. Notation3, Rules.
* Topics: ISO, TMDM, TMRM.
* Topics: Math.
* Topics: HAL / OData / GraphQL / Protocols.
* Topics: Vert.x / P2P / JXTA
* Topics: RxJava / Java 8 / Functional / Streams.
* Topics: Functional JS / OO JS / NodeJS
* Documents: Topics. Tools. Specification.
* Statement:
* (Context : Statement, Subject : Resource, Attribute : Kind, Value : Resource);
* Value Resource according Kind (Roles) Statement Subject occurrence.
* Sets: de aggregated Statements. Reification. Layers (Models) expands Statements for each Kind Attribute.
* Graph:
* (Context : Type, Subject : Node, Attribute : Arc, Value : Node);
* Kind / Type: Aggregated Attribute / Arc occurrences. Type Inference.
* Type Context Arcs Expansion. Augmentations.

Monads:

ResourceURNMonad<ResourceClass : IResourceURN, etc.>

ResourceMonad<ResourceClass : ISubjectResource, etc.>

KindMonad<KindClass : ISubjectKind, etc.> Monad

OccurrenceMonad<OccurrenceClass : ISubjectOccurrence, etc.> Monad

* Proof of Concept: Achieve REST Facade (synchronized) of Relationships given inputs from a system backend:
* Inputs: Aggregate SPO into CSPO: Aggregates Contexts Type / Table / Class Kinds. Aggregate PK Cols, Cols : Occurrence, Val : Resources.
* Inputs (Rel / Graph): (Type / Table / Class, PK : Resource, Col : Occurrence, Val : Resource).
* Inputs (Rel / Graph) FKs: Val : Resource equivalent PKs.
* Input / Canonical: Match Interfaces / Signatures: (Context, Occurrence, Attribute, Value);
* Augmentations / Transforms:
* Data matching. Resource equivalence: identity / comparisons / order transforms.
* Schema matching. Predicates equivalence. Domain / Range types. Order: data flow contexts.
* Behavior matching. Domain / Range values applied functional predicates identity. Order: data flow interactions.
* Encoding / Matching:
* Functional Context:
* Metaclass
* Class
* Instance
* Context
* Occurrence
* Role
* Hierarchy: Roles / Primitives (upper / aggregated hierarchy).
* Input / Canonical: Match Interfaces / Signatures: (Context, Occurrence, Attribute, Value); Attribute / Value Roles in matching interface context. Order / hierarchy encoding.
* Upper Ontology: Need, Product, Good, Purpose. Goal.
* Upper Ontologies: From Primitives to Forms / UI Gestures.
* Units of Measurement (continuos) APIs /  Ontology.
* Discrete (events) APIs / Ontology. Relationships.
* Cube Statements:
* (Fact, Axis, Measure, Value);
* Core Models Statements. Mappings / Transforms Declarations / Applications Reifications. CSPO Functional Categories:
* Core Model Statements: Wrapper Category Types:
* Transform : Mapping : Statement : Resource;
* Mapping Category: Query / Browse.
* Query: (Mapping, Resource, Transform, Resource);
* Transform Category: Assert.
* Assert: (Transform, Resource, Mapping, Resource);
* Example: Query / Assert on following Statement upon CSPO Mappings / Transforms Categories.
* Query: (Distance / Mapping, 1km, Meters / Transform, 1000m);
* Assert: (Distance / Transform, 1km, Meters / Mapping, 1000m);
* Monad Wrapper Role Types Hierarchy:
* Transform : Mapping : Statement : Resource;
* URN : Resource (alignments). Primitives.
* Resource : Root Category. URN : Source / Surrogate Key / Crafted. Naming / Encodings (below).
* Statement : Resource. Wraps Resource as Statement Category.
* Mapping : Statement. Wraps Statement as Functional Transform Declaration Category.
* Transform : Mapping. Wraps Mapping as Functional Transform Application Category.
* Monads Wrapped Roles types / values. Transforms / Mappings. Objects.
* Resource Roles: Reified CSPO Resources types / values, Kinds types / values, Statements, Class, Instance, Occurrence, Occurring, Attribute, Context, Value, Role, Models Roles type / values getters (populated Mappings / Transforms.
* Resources Objects hierarchies / APIs: (Reified Mappings / Transforms): DTOs / Dynamic Functional, DTOs (hashmap) of CSPO roles getters / model domain browsing getters (Functional Transform parameterized Mappings). Flow context: referrers / keys types / values: address::city::street Aggregation / Map. Templates / Mappings: reified declarative Augmentation Data Flows.
* Ontology alignments: Data / Schema / Behavior Augmentations. Model / Schema / Upper / Domains: purposes / gestures (MVC / DCI Mappings / Transforms) layers. Example:
* Transforms: unaEmpresa::unEmpleado::unaPosicion::salary; Salary inferred by context and unaPosicion, unEmpleado, unaEmpresa.
* Class / Instance / Reified (occurrence / ocurring) / Mappings / Transforms:
* Models: Domains / Reified Wrapped Types / Instances. Models, i.e.: Dimensional Domain: Core Model (Transform / Mapping, Resource, Transform / Mapping, Resource) Wrapper Roles, (Dimension, Measure, Unit, Value) Wrapped Instances Types.
* (Distance, 1km, Meters, 1000m); Unit Meters: Occurrence / Occurring Mappings / Transforms Dimensional (upper / aggregated / inferred) Domain Knowledge Assertions Map.
* Statements Augmentation: Dimension::map : Measure, Measure::map : Unit, Unit::map : Values. Wrap C(S(P(O).
* Statement
* Kind
* ContextKind (SubjectKind / ObjectKind flow signature)
* Statement
* SubjectKind
* PredicateKind
* ObjectKind
* Subject
* Predicate
* Object
* Resource
* Context
* Template
* Class
* Instance
* Occurrence
* Value
* Occurring / Context (Statements / Kinds)
* Sets
* Graphs
* Roles (Metaclass, Class, Occurrence, Context, Role)
* Dimensional (Dimension, Measure, Unit, Value)
* Discrete (Relationship, Relation, Kind, Entity)
* MVC / DCI Mappings / Transforms. Example: Forms, Purpose, Gestures, Actors, Roles. Data / Schema / Behavior alignment.
* ESB: Endpoints, Features, Interfaces, Service Process Description / Discovery. Reactive Events Subscriptions. OSGi HATEOAS Endpoints "autowiring".
* BPM: Process, Steps, Flows, etc.
* Augmented Actionable (Process Flows, Items Activation) CMS. Browser: HATEOAS Protocol / APIs / Augmentations. Inferred / Reified / Resolvable Data Flows. Designer: Model Pallete. Declarative core / domains types / instances browsing / discovery "wiring".
* Graph Reified Grammars (upper). Terminal / Non Terminal. Rules / Productions. Mappings / Transform: browse grammar, rules, productions:
* (Rule, Context, lhs, rhs)
* Naming: Kinds / URNs Addressable Encodings. Parsing: URNs Encoded Functional Distributed Resource Resolution. Data Flow Transform / Mappings: Embedded Productions: Augmentations. NLP / NER. Ontology Matching: URN Class Transforms.
* Graph Embeddings: ML Backend Services (ML Predictions Augments Mappings / Transforms). Encodings (Naming).
* Encoding: Deep ML Embeddings. Data: classification, Schema: clustering, Behavior: regression.
* Naming: Auto Encoders. Semantic Hashing. Resources Mappings / Transforms Reified Maps / Tables. Keys / Values Resource Hashing / Resolution Functions: Contextual to Functional Environment State: Mappings Flows / Wrapped State.
* Naming: Augmentations. Contextual Hash Enabled: Functional Mapping Flows Map / Table Encoded / Resolved. Functional Relations: Ontology Matching / Aggregation / Inferences by Hash Encoded Metadata / Transforms Resolutions.
* Clients / Browsers: Peers. Protocol: Reactive Dialogs Prompts. Events. Distributed Data, Schema, Behavior Core Model Statements Encoded I/O: Layers Sync / Augmentation of Knowledge requested from each Peer(s) as Model inputs given resolution of Dialog (Subscriptions) event sourcing state. MVC / DCI Distributed State Transforms / Mappings. Augmented Peer(s) Models: updated View State (flows) / Mappings / Transforms. Rendezvous Peer Role. Local Peer: APIs for local / remote views (MVC / DCI) views (Web, REST) Rendering.
* (...)
* Layers, input layer: Model Roles. Aggregation: Layer Roles shifting until full Layers Roles Statements. Layer: Augments Models.
* Core Statements Roles Resource Interleaved Model (Infer Types, PKs, FKs):
* Occurrence: (Class, Instance, Attribute, Value);
* Occurring: (Class, Instance, Occurrence,  Role);
* Statement Transforms / Relations: order, equivalence, roles, etc.
* Resource of Resource Monad: Occurrence  / Occurrings Quads CSPO Members Aggregation Transforms.
* Resource(t : T) :: contexts :: subjects :: predicates :: objects : Resource(u : U)
* Resource of URNs: Aggregated CSPOs Transforms of Occurrences / Occurrings. Matchings.
* [Resource.of](http://resource.of)(Resource / Class, Instance, Occurring Attribute / Occurring URN, Attribute Value / Occurring Role) :: contexts :: subjects :: predicates :: objects : Resources(CSPO / Resource / occurrence / occurring : URN).
* Resources of Resources: Occurrences / Occurring CSPOs Transforms Matching Wrapped / Wrappings / Transforms Resource Types Shapes Matching Templates. Data flow, apply transforms, order, lists.
* [Resource.of](http://resource.of)(templ : Templ) :: contexts : Resources(c : Context).
* Layer Template Mappings:
* Template : Context : Statement : Resource Layer Roles Monads.
* Layers Quads Aggregation: rotating value role types from previous layer to next layer from V to C):
* Canonical Template Mapping Layers: Aggregation of Template Matching Models Layers until first layer Value is wrapped into final layer Context. Perform Layers Augmentations.
* Monads. Wraps Models Roles. Matching: Patterns wrapped Resource Roles values: placeholders, variables, wildcards, reified instances. Example: Template wraps previous layer Value as Context, Layer Context Value wrapped as Subject value (shifting).
* Core Roles (Wrappers):
* Resource : Monad(x : Resource);
* Statement : Monad(x : Resource) : Resource, Quad DTO.
* Context : Monad(x : Resource) : Statement;
* Template : Monad(x : Resource) : Context.
* Template : Context : Statement : Resource
* Input Layers: Core Statements Roles Resource Shapes: (interleaving graphs models) : Statements CSPO.
* Occurrence Input: (Class, Instance, Attribute, Value);
* Occurring Input: (Class, Instance, Occurrence, Role);
* Output Layer: (Template, Context, Statement, Resource); Input OPSC Resources wrapped in output Layer Roles.
* Sets, Graph, Roles: Services / Augmentations helper Models / Facades.
* DOM Functional Monads Model Traversal / Transforms. Domains Traversal / Transforms due to DOM Values, Kinds and Context Statements, Mappings and Transforms.
* Augmentations:
* Model Augmentations:
* Alignment: Data Matching. Resources.
* Alignment::match
* Alignment::perform
* Alignment::greaterThan
* Alignment::equals
* Alignment::lessThan
* Activation: Schema Matching. Kinds.
* Activation::match
* Activation::perform
* Activation::superTypeOf
* Activation::sameTypeOf
* Activation::subTypeOf
* Aggregation: Behavior Matching. Contexts Flows.
* Aggregation::match
* Aggregation::perform
* Aggregation::beforeThan
* Aggregation::contains
* Aggregation::containedIn
* Aggregation::afterThan
* Domain Augmentations:
* Transforms Reified in Layers Contexts. Pattern Matching Template Layer resolved:
* Mapping::match
* Mapping::apply
* Mapping::Context
* Mapping::Subject
* Mapping::Predicate
* Mapping::Object
* Encode reified Template Mappings / Transforms. Patterns:
* Model / Domain Augmentations Mappings / Transforms:
* (Wrapper, Wrapped, Mapping, Transform);
* Next Layer step: match / apply Augmentations.
* Layers Template: Layer of CSPO Data Flow Contexts Patterns Resolution Resources : Reified (meta) Resources.
* Template, Mapping, Pattern
* (Template, Context, Statement, Resource);
* Resource :: occurrences :: roles :: contexts : Resource
* Context :: Mapping :: Transform :: Kind :: Resource
* Occurrence: (Class, Instance, Attribute, Value);
* Occurring: (Class, Instance, Occurrence,  Role);
* Statement Transforms / Relations: order, equivalence, roles, etc.
* Resource of Resource Monad: Occurrence  / Occurrings Quads CSPO Members Aggregation Transforms.
* Resource(t : T) :: contexts :: subjects :: predicates :: objects : Resource(u : U)
* Resource of URNs: Aggregated CSPOs Transforms of Occurrences / Occurrings. Matchings.
* Resource.of(Resource / Class, Instance, Occurring Attribute / Occurring URN, Attribute Value / Occurring Role) :: contexts :: subjects :: predicates :: objects : Resources(CSPO / Resource / occurrence / occurring : URN).
* Resources of Resources: Occurrences / Occurring CSPOs Transforms Matching Wrapped / Wrappings / Transforms Resource Types Shapes Matching Templates. Data flow, apply transforms, order, lists.
* Resource.of(templ : Templ) :: contexts : Resources(c : Context).
* Monads. Wraps Models Roles. Matching: Patterns wrapped Resource Roles values: placeholders, variables, wildcards, reified instance.
* Context(Statement, Mapping, Behavior)::Kind::Resource
* Layer Template Mappings:
* Template : Context : Statement : Resource Layer Roles Monads.
* Layers Aggregation (rotating role types values for each layer from V to C):
* Layers Aggregation: Perform Layers Templates Matching  Augmentations. Augmentation  Models Statements Metadata producing new potentially matching Contexts.
* (Template, Context, Statement, Resource):
* Canonical Template Mapping Layers: Aggregation of Template Matching Models Layers until first layer Value is wrapped into final layer Context. Perform Layers Augmentations.
* Monads. Wraps Models Roles. Matching: Patterns wrapped Resource Roles values: placeholders, variables, wildcards, reified instances.
* Core Roles (Wrappers):
* Resource : Monad(x : Resource);
* Statement : Monad(x : Resource) : Resource, Quad DTO Statement.
* Context : Monad(x : Resource) : Statement;
* Template : Monad(x : Resource) : Context.
* Template : Context : Statement : Resource
* Wrapper / Wrapped reactive / event driven message bus API.
* Quads I/O: Base Model Protocol. URN Encoding / Content Types. Transforms / Mappings: browse links transforms. Layers: Augments Models (resolve / merge ontologies).
* Implementation:
* Resources: Data aggregated layers statement sets / alignment upper resources. Root Resources facade (Layered Quads IO / transforms: locators, streams, events based activation).
* Metamodels: Domain aggregated layers statement sets / alignment upper resources. Root Metamodels facade. (Layered Quads IO / transforms: locators, streams, events based activation).
* Applications: Interaction aggregated layers statement sets / alignment upper resources. Root Applications facade. (Layered Quads IO / transforms: locators, streams, events based activation).
* Aggregation: compose stack of Resources, Metamodels, Applications IO over layers behavior activations (Resources behavior : Metamodels data, Metamodels behavior : Applications data). Knowledge activation aware resolution of multiple layer parents / children statements IO / aggregation / routes.
* Aggregation: Contexts facade (Layered Quads IO / transforms: locators, streams, events based activation). Browseable / CRUD of Contexts, Interactions, Domains, Data.
* Resources, Metamodel, Application : Context.
* Aggregation: (Application (Metamodel (Data))) : Context (Lang Quad).
* Networking, P2P, ternary routes.
* Dataflow: models, routes: data, operations (shapes, patterns, templates / codata: infer schema / behavior from data / dimensions information and knowledge).
* Dimensions: observations / measures of data.
* Data: instances / occurrences of contexts.
* Contexts: instances / occurrences of roles.
* Interactions: templates / contexts of roles.
* Event / Process. [Schema.org](http://schema.org).
* Dimensional DCI / instance, class, metaclass / templates: definitions / sets (intensional / extensional: representation, identifiers, encoding / sets).
* Naming URI scheme / encoding: dimensional / data / context / interaction hierarchical metaclass / class / instance (occurrence / template) hypermedia addressing. Discovery / browseable (HATEOAS / HAL: REST identifiers, resources, representations, DCI: identifiers encoded / messages). Layers: activation / domain / backends (aggregation, alignment, augmentation) encoding / algorithms.
* Exchange: addressable interaction (class / instance, DCI, dimensional, layers). Routing algorithms / encodings.
* Semantic tagging and search in browse sessions / threads rels. Paths. Discovery. Referrer / refs rels.
* Semiotics: Object: Resource, Concept: Class, Sign: Representation. Class / metaclass: occurrences / templates.
* Semiotics: Syntax: Data, Semantics: Context, Pragmatics: Interactions.
* Event: State change (dimensional measure in context axes), period. Class, metaclass, instance rels / templates. Data: Dimensional DCI, Information: Events, Knowledge: Data / Information aggregation, inferences, predictions (augmentation / activation).
* Activation: aggregation, alignment, augmentation, reasoning, rule, flow, schema, data inference / materialization (DCI / data, information, knowledge layers). Dataflow: ontology driven routes, patterns / ops streams (feedback), ontology (domain backend schema / DCI) templates, behavior (functional IO) events.
* Encoding: nodes (ops routes, pattern shapes: signatures), data (routes, pattern shapes: dimensional addressing) declaratively stated, dataflow bindings (discovery). Nodes as data, data as nodes: functional resources (dimensional DCI: extract signatures / addresses).
* Data, schema, behavior blend / merge / compose. Discover alignments, contexts, roles.
* Definitions, blending, loading. Frontend. Use cases. QA, Process assistant, systems integration (flows): purpose fulfilment (steps, roles, items: DCI / dimensional aggregation).
* Embeddings: encoding, semiotic ops (resource nodes / arcs: dataflow signatures / patterns).
* Sets: encoding, bitstring, predicates (order: comparisons). Convert 0/1: divide measure by possible values count. Ternary: comparators. Quads encoding: convert to aRGB / IPv4 address (routing ternary masks). Tensor shapes: recursive quads. Routing / dataflow: dynamic templates node (ops), arc (data) resources 'activation' signatures.
* Lamda WS: code representations. Resource IDs: (representations, verbs templates / contexts, types).
* Metamodel Resources: URIs, representation, messages (instance / occurrence of resource in context / dimension from browsing / referrer state, streams). REST HATEOAS / HAL by message schema resources metadata (browsing rels / state reified referred dimensional message DCI resources). Message resources: referrer / rels (resource / context). Resources as nodes / arcs (code / data) dataflow (discovery, signatures / template, patterns). Layers.
* IPv6: Resources, messages, routing / encoding (HTTP / REST / Protocols / P2P / data, schema, behavior routes: activation) as addresses (quads). Layers. Resources occurrences / messages addressing (dataflow node / arc signatures / template patterns, contexts). Message resources encoded in addresses. Addresses: encoded resources (messages / dataflow rels, ops, data).
* Resources / messages: encoded as addresses. Interactions / contexts / templates.
* Encoding, address: resource IP / port / index res. Paths: rels / arcs / roles / contexts / occurrences (other resources w./ paths, commands w./ templates, contexts).
* Beans serialization: declarative XML / IDL. Templates (commands / data). Serializable / externalizable (transforms). JAF / JCA. CDI / injection / bindings.