# 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.

# Encoding

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:

## Encoding Layers

### Addressing Sail

* 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.
* 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\*.

### Matching Sail

* 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.

### Persistence Sail

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

## Encoding Layers Object Model

### Serialization

* DagCBOR. XML (Beans, Externalizable): Functional Roles (metaclass, etc) Functional Transforms (Aggregation Templates). HAL / JSON: Functional Fields (codat / data flow).

### Model

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).

# Augmentations

## 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).

### Aggregation Sail

* Aggregations: ResourceURNs Source IRIs Sets / Layers streams / events (Resources, Occurrences, Kinds) parse / Occurrences population.

### Alignment Sail

* Alignments: Aggregation traversal: ResourceURN URN IDs Model population. Merge / Matching, order / relations / contexts. Encoding (methods).

### Activation Sail

* Activations: Relationship Models I/O (DCI Layers / expanded SPO Aggregations feedback). DIDs URN hashing / generation (HATEOAS Endpoints). Data Flow.

## 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).

# 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

### 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

Services Layer Functional Dynamic Object Model:

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.

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)

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);
* 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);