* Aggregation: Statements layout. Occurrences.
* OrderKind Statements.
* Template Population.
* Augmentations:
* Alignment:
* Data / Schema Matching. Layer API: Statements, States, Kinds, Resources.
* Data. Model.
* Model Resources / Kinds / States / Statements Aggregation.
* Templates: Kinds / States hierarchy orders Statements Resources.
* Schema matching. Predicates equivalence: Domain / Range types / values. PredicateKinds of same SK / OK (Relationship).
* Activation:
* Flows (Mappings) / Transforms Behavior Matching. Layer API: Transforms, Mappings, Statements,Templates.
* Schema. Context. View.
* Template Statement Flow matching Template Kind  Statements (Dataflow Mappings) Mapping Kind Resource populates Transform Relation Statement Kind Resource. Aggregate Template Statements Mappings Transforms (State / Kinds flows inferred / ordered). Relationship Mappings: PredicateKinds of SKs / OKs (Kinds Roles flows inferred / ordered).
* Aggregation:
* Forms: Relationship (Dimensional / Discrete) DOM API Aggregated Mappings (Transforms). Layer API: Relationship, Relation, Role, Occurrence, Context (DCI Facade).
* Behavior. Interaction. Controller.
* Mapping Statement Transform updates DCI Model Relation Roles. Dataflow Forms subsequent Mapping Forms.
* 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. Populate Templates.
* Kinds: Aggregate Attributes.
* State: Aggregate Kind Attributes Values.
* Hierarchy: Kinds Attributes set (super) subset (sub) Kinds relationship.
* Order. Aggregation: Kinds / States lattice / tree. Populate Templates.
* Domain Model Object Hierarchy:
* ClassName :: (aggregatingClass, subject / instance, attribute / predicate, value / object);
* Classes (Sets) domain hierarchy:
* OntResource
* Subject : OntResource
* Predicate : OntResource
* Object : OntResource
* Kind : OntResource
* SubjectKind : Kind
* PredicateKind : Kind
* ObjectKind : Kind
* State
* Statement : Kinds, SPOs
* Templates : Kinds, SPO
* Mappings : Kinds, SPO
* Transforms : Kinds, SPO
* Functor / Category: Resource Monad (of OntResource hierarchy). Dynamic typing DOM / DTOs Kinds members.
* Resource<T extends OntResource>::of(T extends OntResource);
* OntResource: Uniform Resource domain category interface:
* getSet
* getKind (in Statement occurrence functional context).
* getOntResource (Kinds get SPO in Statement occurrence functional context).
* getResource (this)
* getContext
* getOccurrences : Type static instances list
* getAttribute
* getValue
* getQuadContext
* getQuadSubject
* getQuadPredicate
* getQuadObject
* unit / join / bind : Resource<T extends OntResource>
* Map, flatMap, composition. Dynamic functional types / transforms: Kinds.
* Domain Object Members, i.e.: getSubjectKind.
* OntResource:
* (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:
* (State, SubjectKind, Predicate, Object);
* PredicateKind (PK) : Predicate. Subject / Object Intersection:
* (State, Subject, PredicateKind, Object);
* ObjectKind (OK) : Object. Predicate / Subject Intersection:
* (State, Subject, Predicate, ObjectKind);
* State : Kind, SPO:
* (Template, Resource, Attribute, Value);
* Order Kinds: Parent, Previous, Sub, Next, Equals Context Resources (SPO) Kinds. Populate Templates: Resource in Statement (Kind) OrderKind Kind. Statements Order in relation to Resource axis. OrderKinds: Equals equivalent type, sameAs: singleton Kind.
* Template : Kind, SPO:
* (Statement, Resource, OrderKind, Kind);
* Statement : Kind, SPO:
* (Mapping, Resource, Resource, Resource);
* Mapping : Kind, SPO:
* (Transform, Statement, Kind, Resource);
* Transform : Kind / SPO:
* (Relation, Statement, Kind, Resource);
* 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);
* 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: Comparison. A / B < 0: A < B. A, B indexes: hierarchy tree positions.
* Model API / Encodings:
* Model: OntResource. DOM.
* Encoding:
* Resource Monad, Resources and Functional Context: REST Verbs in State, Forms, Flow Relationship DOM Context. Uniform Representation: Render Context state: Forms, Relationship (Dimensional / Discrete) DOM API. Layer API: Relationship, Relation, Role, Occurrence, Context (DCI Facade).
* Sets Resources REST HATEOAS / Reactive / Functional Streams Data Flow IO Model Statements:
* Functional Data Flow:
* Functional Context: Arrange Model Statements in a uniform Resource roles based interface: CSPO, relations (Aggregation, occurrences, roles, etc.).
* (Resource, Kind, State, Template, Statement, Mapping, Transform, Relationship, Relation, Context, Role, Occurrence);
* URNs Encoding.
* 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.
* Features:
* ESB. BPM. CMS. NLP. BI.
* Naming: URNs.
* 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.
* Encoding / Matching:
* Functional Context:
* Metaclass
* Class
* Instance
* Context
* Role
* Occurrence
* Hierarchy: Roles / Primitives.
* Upper Ontology: Need, Product, Good, Purpose. Shapes.
* Upper Ontologies: From Primitives to Forms / UI Gestures.
* Units of Measurement (continuos) APIs /  Ontology.
* Discrete (events) APIs / Ontology. Relationships.