* Encodings:
* Relationship / Measures: Discrete / Dimensional.
* Propositional Calculus:
* S: works(John, aJob);
* P: EarnsMoney(Employee, Job);
* (intermediate / auxiliary assertions)
* O: earnsMoney(John, johnsSalary);
* Relation Kinds. TBox, ABox. Order, reflexive, transitive, symmetric. Relation Type / Instance declaration encoding, assertions.
* Encoding. Models. Inferences Resolution Lattices.
* Models: Events (Statement\* Message) I/O, Synchronization. Dataflow semantics.
* Model: Plain RDF CSPOs.
* Services:
* Augmentations: Events Dataflow
* Model: Sets. Resources, Kinds, Contexts. Registry (ISubject, Resource<Subject>). API.
* Service: Aggregation. Registry. Controller. Available Contexts Interactions.
* Model: Resource Kinds Occurrences. Naming (ResourceURN). API
* Service: Alignment. Naming. Model. Available Interactions Data.
* Model: Dimensional FCA Lattices. Key Value (Map Reduce) inferences. Quad (octal) digits in object attributes values (Statement CSPO / Set attribute position). Index. API.
* Service: Activation. Index. View. Key Value. Available Data Contexts.
* Functional Contexts: Predicates and reified APIs transforms (as Resources) Resources traversal (R) / transforms (CUD). Domain / Range Dataflow. Streams / Workflows / Templates. HAL / HATEOAS Events semantics.
* Events Dataflow. MVC / DCI OGM DOM DDD APIs. Event Messages HATEOAS: Services / Events driven Protocols / Templates
* Input:
* Encode / Match / Augment Resource Data.
* Browse / Augment Index for Resource (Data) Available Contexts.
* Query / Augment Registry for Available Context Data Interactions.
* Match / Augment Naming for Context Data Interaction Data.
* Output:
* Interaction Data matches Index for further available Contexts.
* Steps (layers) streams driven by a HATEOAS / HAL Protocol.
* Messages Predicates encode Dimensional (FCA Lattices) Contexts Values (available / follow dimension values)
* Services APIs: Connectors / Facades: Events / Protocols. Message Templates: I/O. Retrieve: Pattern, CUD: Statement which has unknown S/O for Predicate.
* FCA Inferences. ISO TMRM / TMDM / TMCL / TMQL / HyTime.
* APIs: RDF / RDFS / RDF\*  OWL / OWL\* / SPARQL / SHACL / ShEx / Inferences / Formats / Syntaxes (inferences/augmentation/resolution models encoding: URNs lattice):
* FCA Predicates Contexts (S, p, O). P(S, O). Schema defined models. Verbs definitions: domain / range dataflow.
* FCA Scaling: Resources, Kinds, Contexts Functional Categories levels. Reification(\*). Transform. Traversal.
* FCA DCI: Metaclass, Class, Instance, Context, Role, roles Occurrences scaling lattice
* FCA URNs: See Sets Model (Resources, Kinds, Contexts) Functional API:
* schema:[context]/[ctx:subject]/[ctx:predicate]/[ctx:object]#[hashResource]?[queryString]
* [bodyResource] : HTTP Headers. Content-type, content / body. Encode Graph from navigation / traversal metadata.
* [ctx:resource] : Lattice Context plus concept occurrence.
* FCA: Contexts(Object, Attribute); Nested Contexts. Scaling.
* Resources: S(Predicate, Object);
* Resources: P(Subject, Object);
* Resources: P(Object, Subject);
* Kinds: Scaling (aggregates resources).
* S(P, O); SK.
* P(S, O); PK.
* P(O, S); OK.
* Contexts: Scaling. Aggregates Kinds.
* SK(PK, OK); Statement.
* PK(SK : S, OK : O); Mapping.
* PK(OK : O, SK : S); Transform.
* Inference by analogy. Model : Posible functional inferences: Datatypes, (upper) schema, instances (Sets / Kinds) domain / range.
* Dimensional (Dimension, Measure, Unit, Value). Unit: domain / range.
* Relationships (Relationship, Relation, Role, Entity). Role: domain / range.
* (Dimension, Event : Measure) (Functorial Inference Dimension / Relationship) (Dimension, Event : Measure);
* Bookmarks / Bibliography / Tools.
* Domains:
  + BI. Pentaho.
  + ERP. Tryton.
  + DDD: Instant API Backends
  + Social / Purposes: Solid / StratML.
* Features (RDF4J Sails):
  + Components:
  + Connectors: Traits. CDI Bus Signatures / Protocols (Events Encoding)
  + MDM: Onto Merge Matching. Traceability / Graphs Traversal. Models Bus.
  + ESB: Integration: Connectors Bus.
  + Rules / Inferences.
  + Workflows.
  + TMDM, TMRM: ISO TopicMaps.
  + FCA: Formal Concept Analysis.
  + Sails stack: from plain RDF / RDFS / OWL / Sem Web stack inferences through Augmentation Sail(s) to DDD Runtime: OGM / DCI HATEOAS Applications.
  + Sails stack: from plain RDF / RDFS / OWL / Sem Web stack inferences through Augmentation Sail(s) to DDD Runtime: OGM / DCI HATEOAS Applications.
* Augmentation Sails:
  + Alignment: Naming. Resources Model. Available Interactions Data.
  + Aggregation: Registry. Controller. Sets Model. Available Contexts Interactions. Dataflow.
  + Activation: Index. View. Key Value. Available Data Contexts.
  + Naming: Resources Model. Model. Available Interactions Data.
  + Naming: URNResources. Encoding / Resolution: FCA / TMRM Concept Lattice (nested key value bitstrings). Model. Available Interactions Data.
  + Registry: Queries. Sets Model. Controller. Available Contexts Interactions.
  + Registry: Hierarchical Key Value (FCA / TMRM) store. Events Sourcing. Controller. Available Contexts Interactions.
  + Index: Hierarchical Key Value (FCA / TMRM) store. Events Sourcing. View. Available Data Contexts.
  + Index. Queries. Model : Inferences : Schema. Sets: Templates (Contexts). TMDM. Graph (Spark) ANN / key-value aggregation / mappings. View. Available Data Contexts.
  + Encoding / Resolution: FCA / TMRM Concept Lattice (nested key value bitstrings). Event Sourcing: Lattice order relation.
  + Dataflow:
  + Input:
  + Encode / Match Resource Data.
  + Browse / Augment Index for Resource (Data) Available Contexts.
  + Query / Augment Registry for Available Context Data Interactions.
  + Match / Augment Naming for Context Data Interaction Data.
  + Output:
  + Input Steps Dataflow resolve streams aggregated in a fan in / fan out fashion. Functional schema / domain transform / mappings / inferences applied.
  + Browser Extensions. Clients Connectors.
  + Deployment Connectors: Google Apps. Solid. DIDs.
  + Runtime: OGM / DCI: OpenRDF Elmo. Bus Endpoints. DOM HATEOAS.
  + Runtime: Qi4j (RDF Entity backend). Sesame. Bus Endpoints. DOM HATEOAS.

**Sets Aggregation Object Models**



* 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>
* 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.