* ActivationService:
* Referrer.
* Address.
* Body.
* Next State.
* (ActivationContext : Relation / Referrer, Statement : Body, Kind : Address, Resource : Next State);
* ActivationService Resource Object (Resource) is next / matching Resource for request context (Referrer, Address, Body).
* Activation Patterns (DOM Layers hierarchy matching / Layers reified Resources matching):
* (Resource, Role,
* (Kind, Resource, Role,
* (Statement, Kind, Resource, Role);
* (Relation, Statement, Kind, Resource);
* (Entity, Relation, Statement, Kind);
* (Relationship, Entity, Relation, Statement);
* (Flow, Relationship, Entity, Relation);
* (Domain, Flow, Relationship, Entity);
* Semiotic / Dimensional Roles Facets: Idem for equivalent Relation hierarchy layer.
* Monads: Reify available Transforms as activable Resources (Function addresses). REST / HATEOAS HAL.
* Prompts / Dialogs: Function arguments (values / options) shown as link addresses in Transforms navigation Flows. Activation browse of Resources in Transform context.
* Chained Activations for complete contexts resolution / flows. Complete layers productions rendering / navigation from higher to lower layers.
* Workflows / Prompts Model Relations / Services:
* Domains. Goals. Profiles. Product / Good / Need. Roles.
* (Domain, Flow : Flow, Role : Relationship, Item : Entity);
* Domain<Flow[]>;
* Flow: Use Case. Produced / available for Domains. Role: Prompts. Produces / available in state of Flow. Item: Product / Good / Need (Goal). Produces / populated by Roles (Prompts).
* (DomainService : Relation, Resource : Domain, Context : Flow, Item : Relationship);
* Workflows (domains goals) framework:
* Workflow kind: Need, Good, Product. Roles. Inter-domain workflows.
* Prompts: context wiki (domain learning roles, domain evaluations: skills).
* Prompts: Resource matching domain skills answers flows: values / decisions.
* Prompt: value / decision query. User / Service response.
* Workflow tray: declarative API interface (workflow concepts). Guided assistant / learning.
* App: questions with a purpose. Domains roles / skills filters.
* Matching. Distributed Addressing Encoding:
* Encoding: Encode metaclass, class, instances, occurrences (contexts) in addresses.
* Matching: Same objects resolve to equivalent addresses in different models.
* ResolutionService (Matching):
* (ResolutionService : Relation, Resource : Entity, Model : Relationship, Resource : Flow);
* Meta Meta Model: (Relation, Kind : Entity, Statement : Relationship, Resource :  Flow);
* Resource<OntResource[]>; Aligned Resources (metaclass, class, instance, role, occurrence);
* Kind<Resource[]> : Aligned Types.
* Statement<Kind[]> : Aligned Statements.
* Relation<Statement[]> : Aligned Assertions.
* Entity<Relation[]>;
* Relationship<Entity[]>;
* Flow<Relationship[]>;
* Domain<Flow[]>;
* Service Resources. Context scoped prediction generalizations (encodings):
* (PredictionService : Relation, Context: Entity<PredictionService>, Features : Relationship<Entity>, Output : Flow<Relationship>);
* Naming Service (synsets, generalization/specialization term rels):
* (NamingService, Context, TermRel, Term);
* Registry Service (hierarchical key / value):
* (RegistryService, Context, Key, Value);
* Index Service (Apache Lucene, Vector Space Model Triple / Quad polygon encoding):
* (IndexService, Context, Term, Result);
* IOService (Connectors):
* (IOServiceConnector, ContextResource : Entity, Attribute : Relationship, Value : Flow);
* ResolutionService (Matching):
* (ResolutionService : Relation, Resource : Entity, Model : Relationship, Resource : Flow);
* Entity: Aligned Resource.
* Relationship: Aligned Schema.
* Flow: Aligned Behavior (schema resource data flows).
* FCA Contexts from Sets aggregation:
* Set Roles: Context, Subject, Predicate, Object.
* Sets aggregation: Statement, Kind (SuperKind, Kind, Attribute, Value) Attributes: class / Values: metaclass, Resource (Meta Model Roles: Kind context, Resource SPO), Context (Relation). Reified Kinds.
* Aggregation streams: Sets reactive events aggregation. Sets (ordered) description APIs.
* Sets aggregation: FCA Contexts scaled objects / attributes from Sets aggregation. FCAAPI.
* Ontology Matching:
* Data, Schema, Behavior matching / alignment.
* Data: keys / values.
* Schema / Information: relation tuples rows.
* Behavior / Knowledge: relation tuples rows data / information flows (dimensional).
* FCA Augmented Resources.
* Relationship Monad (aligned data):
* Relationship<Relation[]> Monad. Relation[]: selector, Relations which are instances wrapped by this Relationship scope.
* ToDo: Relation selector API.
* Relationship Monad (instantiated from Relations) functions: assert(e1 : Entity) : Function<Entity, Entity>;
* ID Relationship Monad:
* Relationship Monad instance wrapping Relations of selected (predicates) identical objects (Resources).
* Relationship Monad assert dispatch:
* Invoke Entity assertion matching logic against each individual Relationship Relation and collects results.
* [anEntity.flatMap](http://anentity.flatmap)([anId.assert](http://anid.assert)(e1 : Entity)) : e2 : Entity (anEntity if same Entity, previous / next Entity if not same Entity).
* ID previous / next Entities in Relationship Relation complimentary (Entity comparison complements) axes.
* Entity Monad built in Relationships: ID, equals, inverseOf, parent, child, previous, next. Apply Relationship assert in the same manner than ID. Logical browsing. Streams (Relationship ordered Entity iterators).
* Domain Models Entities / Relationships: transforms of underlying Entities given Relationships contents.
* Relationships (upper domain): before, during, after, cause, effect, implies, etc.
* Relationsip assertions reified / parsed as / from Relation Statements (Messages statements predicates).
* Entity Relationships rendered / parsed as / from Relations, Kinds, Statement, Resources (Message contents).
* Domain assertions:
* [anEntity.flatMap](http://anentity.flatmap)([aRelationship.assert](http://arelationship.assert)()) : (e2 : Entity) : Select Entities matching Relationship in anEntity (i.e.: retrieve employments).
* [anEntity.flatMap](http://anentity.flatmap)([aRelationship.assert](http://arelationship.assert)(e1 : Entity)) : (e2 : Entity) : Assert Relationship Entity e1 in anEntity (i.e.: append employment).
* [aRelationship.remove](http://arelationship.remove)(e1 : Entity);
* Entity Monad (aligned schema):
* Entity<Relationship[]> Monad. Relationship[]: selector, Relationships which are instances wrapped by this Relationship scope.
* Flow Monad (aligned behavior):
* Flow<Entity[]> Monad. Entity[]: selector, Entities which are instances wrapped by this Flow scope.
* Meta Meta Model:
* DOM: OntResource, Object, Occurrence, Role. Express Meta Model levels metaclass, superclass, class, instance, occurrence relationships.
* (Superclass, Class, Occurrence, Instance);
* Metaclass: Role.
* Occurrence: Role instance.
* Meta Meta Model:
* (Relation / Model : Kind, Kind, Statement, Resource);
* Meta Meta Model Models (Meta Models):
* Semiotic Meta Model:
* (Context, Object, Sign, Value);
* Model example:
* Dimensional Meta Model:
* (Dimension, Unit, Measure, Value);
* Model example: time, events, cause / effect.
* Dataflow Meta Model:
* (Augmentation, Mapping / Predicate, Transform / Template, Resource);
* Model example:
* OntResource
* Object
* Occurrence, Object
* Role, Occurrence, Object : CSPO hiers Sets.
* Resource : Role Set member.
* Statement : Set members Role aggregation.
* Kind: Statement Resources aggregations. Roles intersection sets.
* Relation: Kind Statements aggregations. Transform: Kind Resources related to themselves (ID Rel), then Relations to other Resource via Dataflow Kinds domain / range relationship (ordered).
* Relation: aggregated aligned entities. Views (transforms). Kind members occurring in Statement Resource(s). Functors / Monads:
* Relationship<A : Relation>::flatMap(F : Function<A : Relation, B : Relation>) : Relationship<B : Relation>;
* Entity<A : Relationship>::flatMap(F : Function<A : Relationship, B : Relationship>) : Entity<B : Relationship>;
* Function: declarative dataflow transform.
* Dataflow Kinds domain / range: Grammar. Reify Kinds as CSPO and assert Statement. Aggregate further Kinds (until primitives).
* Valid Statement (Grammar / Relation): domain / range, CSPOs backing assertions apply.
* Reactive streams / event sourcing / driven:
* I/O: Parse / aggregate input Statements into corresponding Roles / Resources. Aggregate / match Kinds. Relations: render / activate. Resolve output Statements.
* Built in Relation(s): ID, equals, inverseOf, parent, child, previous, next, etc. (upper ontology / meta model). Composites: Monad Zippers.
* Tools:
* NakedObjects / Apache Isis. Apache MetaModel. JBoss Teiid. JDBC. JCA. Apache Stanbol. Apache Clerezza. OData / OpenAPI. JSON-LD. Spring HATEOAS / HAL. Apache Any23. D2RQ. R2RML.
* Backend Architecture (ToDo):
* Models. Messages. Encoding. Endpoints. Protocol. Domain Connectors. APIs.
* XML: XSL, XPath, XLink, XPointer, XQuery, XForms: hypermedia addressing / state / flows encoding / Message endpoints protocol.
* Models: Dispatch to Model layers context resources streams. Message IO. Endpoint Message matches in Model context: activation (Dataflow).
* Messages: Model layers scoped context statements.
* Encoding: Model layers scoped context statements.
* Endpoints: Model layers context resources streams (pub / sub topics). Topics: OntResources(s). Signatures: Dataflow Message IO wiring.
* Protocol: Message Dialog: i.e.: XML encoded Context statements Message IO with Model layer scoped prompts, placeholders, wildcards, variables. Models. Messages. Streams.
* Domain Connectors: Tools. Service Resource: URL, streams (Messages I/O).
* Message Matching:
* FCA Augmented Models: Context objects / attributes: layer quad resources Role, Kind, Resource.
* Resource Context Concept: Resource x Kind.
* Resource Context Object: Role x Resource.
* Model / Schema Matching: FCA Resource Context Concept.
* Data / Resource Matching:
* FCA Resource Context Object.
* Matching Productions:
* Result Set (query / augmentation result).
* Augmentation (aggregation of new statements, alignment of new knowledge, activation of transforms / flows: result set).
* Relationship / Entity monads mappings results.
* FCA Scaling: Role > Kind > Resource aggregation of matching objects / attributes.
* FCA Augment Resources: Role, Occurrence, Object (Resource), Concept, Object, Kind. Grammars. Match schema, instances. Mapping transforms: match behaviors.
* APIs:
* Contexts (DCI / HAL / HATEOAS):
* Context Guided Data augmented (contextual hypermedia) Interactions. Wizards APIs. XForms: rendering (REST HATEOAS).