* Introduction: Domain Driven Functional Reactive Dataflow (Mapping Signatures) Services Alignment and Integration.
* Implementation:
* Connectors: Reactive Streams / Message Based ESB. Custom Domain / Platforms Services Implementations. ServiceMix / OSGi / Spring / Vert.x.
* OntResource / Resource Domain. DOM. Sets. Models. Encodings. APIs. Layers (Augmentations): RDF4J SAILs.
* Endpoint: Spring HATEOAS. Augmented Browser / Console. Plugins.
* Order. Augmentations:
* Alignment: Aggregate Resources / Kinds / Statements into type hierarchy. Ontology Matching / Merge. Data. Model. Value Matching. Attribute / Value Aggregation: States. (Statements).
* Activation: Aggregate Kinds hierarchy tree. Build Resources / Kinds / Statements / Mappings / Transforms. Schema. Context. View. Type Inference: Attribute Matching / Aggregation. Attribute / Value Aggregation: Flows. Forms. (Mappings).
* Aggregation: Bind Resources / Kinds / Statements / Mappings / Transforms Statements Contexts. Behavior. Interaction. Controller. Attribute / Value Aggregation: Events. (Transforms).
* States: Data. Model. Kind aggregated Statements. Retrieve States Forms Mappings.
* Flows: Schema. Context. View. Mappings. Forms from Mapping matching aggregated Statements Kinds. Bind Form State in available Kinds bindings (Mapping Kinds: State fields).
* Events: Behavior. Interaction. Controller. Transforms. Bind Mappings matching aggregated Statements Kinds into matching Transforms Kinds. Perform Transform: update DCI Context Model. Update Statements (Transform Kind / Resource set / get: fields.).
* DCI Relationship Contexts: Data / Schema / Behavior Model.
* Sets (Resources, Subjects, Predicates, Objects, SubjectKinds, PredicateKinds, ObjectKinds, Statements: Templates / Mappings / Transforms) abstraction for representing Augmented RDF Graphs.
* (SubjectKind, Subject, Predicate, Object);
* (PredicateKind, Subject, Predicate, Object);
* (ObjectKind, Subject, Predicate, Object);
* (Statement / ParentKind, SubjectKind, Predicate, Object);
* (Statement / ParentKind, Subject, PredicateKind, Object);
* (Statement / ParentKind, Predicate, Subject, ObjectKind);
* Statement: (Kind, Resource, Resource, Resource);
* Mapping: (Transform, Context, Statement, Kind);
* Transform: (Context, Statement, Kind, Resource);
* DCI Context Model:
* Class: Relationship (PredicateKind, SubjectKind, PredicateKind, ObjectKind);
* Metaclasses: PredicateKind SubjectKind / ObjectKind.
* Context : (Relationship, Statements / Context, Role, Occurrence);
* Role : (Context, Occurrence, Metaclass, Resource);
* Occurrence : (Role, Context, Relation, Instance : Resource);
* Relationship 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);
* 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.
* 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.
* Type inference: Implement recursion, aggregation, order, data flow, activation, alignment. Functional streams.
* 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);
* Encoding:
* Sets Resources REST HATEOAS / Reactive / Functional Streams Data Flow IO Model Statements:
* (Transform, Mapping, Statement, Kind);
* Functional Data Flow:
* Transform::Mapping::Statement::Kind::Resource;
* Resource::Kind::Statement::Mapping::Transform;
* URNs Encoding: Type hierarchy / occurrences.
* Order: Comparison. A / B < 0: A < B. A, B indexes: hierarchy tree positions.
* 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.
* Reified Kinds. Reified Roles.
* Shapes. Templates. Kinds Members Kinds.
* Occurrence: (Class, Instance, Attribute, Value);
* Occurring: (Class, Instance, Occurrence,  Role);
* 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.: [rsrc.occ.role.rsrcs.ctx.roles](http://rsrc.occ.role.rsrcs.ctx.roles). Resource Roles getters.
* Data layer: matching alignment transforms. [rsrc.type.rsrcs](http://rsrc.type.rsrcs). Resource / Services augmentations.
* Reified Mappings / Transform. Template Patterns Roles populates corresponding Quad item with matching source Resource.
* Alignment: Discrete / Dimensional frontend aggregation / expansion APIs. Behavior Matching Templates.
* Shapes: "uncle".
* Templates: reify Transforms. Aggregate Template Statements: mappings Resource value domains, predicates, ranges. Matching / Dataflow.
* Template Statement: Statements aggregates Kinds Occurrences for Context 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.
* 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.
* 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.
* 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.: [rsrc.occ.role.rsrcs.ctx.roles](http://rsrc.occ.role.rsrcs.ctx.roles). Resource Roles getters.
* Data layer: matching alignment transforms. [rsrc.type.rsrcs](http://rsrc.type.rsrcs). Resource / Services augmentations.
* 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.
* Upper Kinds Hierarchies of Metaclass, Class, Instances, Contexts, Roles, Occurrences Meta Model Primitives in Layers Contexts Statements.
* Upper Ontologies: From Primitives to Form Gestures.
* Canonical CSPO Statements Reification / Rendering (aggregation / expansion) of Type / Subject Statements.
* 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).
* Functional Context:
* Metaclass
* Class
* Instance
* Context
* Role
* Occurrence
* Hierarchy: Roles / Primitives.
* Upper Ontology: Need, Product, Good, Purpose
* Upper Ontologies: From Primitives to Forms / UI Gestures.
* Units of Measurement (continuos) APIs /  Ontology
* Discrete (events) APIs / Ontology
* Data, Schema, Behavior Layers Meta Model Roles: (Metaclass, Class / Instance, Context / Role, Occurrence) as  Resource Occurrence / Role Attributes / Values.