* Profile App:
* Social exchange network. Purpose driven assets management and Collaboration Tool.
* Metaclass, Class, Instance, Context, Occurrence, Role Resource Metada.
* Asset, Need, Good. Purpose.
* Role (discrete) relationships. Translation (actors context) entailments.
* CSPO, Kinds. Statement, Context, Mapping quads. Sets entailments.
* Dimensional (continuous) Relationships. Translation (contexts) entailments.
* Measure / Value (Dimension / Unit) Pairs. FCA Contexts (scaling) inferences entailments.
* Protocol: Semantic Identifiers. URNs.
* Protocol: Context Driven Interaction REST P2P (SIDs URNs: Resources  DCI Dialogs). Runat peer resolution addressable / browseable Messages interactions: request / response Message streams DCI dialogs. Embedded session semantics: event sourcing / history terms / roles resolution / navigation).
* Messages: SIDs URNs Case Classes Statements. Statement Data Pattern Matching. State Flows: Reactive Events Messages.
* Protocol: SIDs URNs Resources. Endpoints: Case Classes Aggregated Message Signatures, Aligned Statements Data Pattern Matching Message Events Resource Statement Occurrences. Resource Monad.
* Core Model Upper Resources (DCI Context / Facets: Metaclass, Class, etc. as Resource, root navigation Context Resource). Aggregation (schema cases) / Alignment (resource statements occurrences): Activation.
* Core Model Functional Transforms: Functional Activation Statements:  Aggregation Schema Case Classes Statements / Alignment Message Events Resource Statement Occurrence.
* Protocol: GET URN Case Classes (Aggregation) / Statements Data (Alignment) Message Events Resource Statement Occurrences.
* Protocol: GET Browse Resource Aggregated / Aligned Message Events Resource Statement Occurrences. Build Context State Flows (Monad Functional Activation).
* Protocol: POST URN Navigation Context State Built Resource Activation Data Statements.
* Protocol: POST Subsequent entailed Context Browsing / Events Functional Transforms Activations.
* Monad: Resources (Metaclass, Class, etc.). Context.
* Transform: Statements (schema and occurrences).
* Sample Workflow:
* aResource.flatMap(anStatement) : aResourceOccurrence;
* aResourceOccurrence.flatMap(Activation::KindsCase) : aKindResource;
* Activation::[Role]OccurrencesCase \*: Occurrence[Role][];
* DCI / MVC DDD Application Layer: OGM (Sesame Elmo / Alibaba. Qi4j). Core / Domains ontologies.
* Resource: Types hierarchies / instances / occurrences (URN, Statement, CSPORole, Kind).
* Relationships (discrete / continuous). Order. Translation / Equivalences entailments.
* Metaclass, Class, Instance, Context, Occurrence, Role Resource Metadata Maps Monad with contextual CSPOs Statements (schema and occurrences) for Resources in Roles.
* Resources: John, Peter, Mary, loves, friendOf, loverHasFriend.
* Transform / Mapping: John :loves Mary;
* Transform / Mapping: Peter :friendOf John;
* Transform / Mapping (Expanded Knowledge): Mary :loverHasFriend Peter;
* Browse Resources: Functional Activation: Transforms / Mappings Contexts Knowledge Expansion. Idem for Kinds and Schema Statements Aggregations / Alignments.
* Expanded Knowledge: Concrete and Navigation Context entailed / materialized Schema and Occurrences Statements.
* Functional Relation Predicates:
* FCA / TMRM:
* Lover(John, Mary);
* Loved(Mary, John);
* Love(Lover, Loved);
* State(Action, Passion);
* Action / Passion: Resource / State Roles.
* State: Context (Action / Passion Pairs. Verbs).
* Resource Roles: Monads.
* State Roles: Mappings / Functional Transforms (Stateful Contexts Browsing).
* Case Classes: Verticles Event Bus Pattern Matching (Dispatcher / Signatures).
* Reactive / Event Driven: Verticles DIDs (Distributed IDs) distributed patterns routing registry. Resource / Applicable graph logs. Rx Facade.
* Pattern Matching: Case Matching CSPOs, Kinds Types / Instances Matching. Yields corresponding Monad Type / Instance Wrapper Verticle.
* Pattern Matching: Case Matching Statement, Mapping, Transform Types / Instances Matching. Yields corresponding Signature Function Verticle.
* Monads Verticle Endpoints (topic) consumes Monads, produces available Functions in Monad Functional Context,
* Monad Consume Cases: Instantiate Function Verticles.
* Function Verticle Endpoints (topic) consumes Functions in Monad Functional Context, produces Monads available for Function Application.
* Function Consume Cases: Instantiate Monad Verticles.
* Monads:
* Resource
* SPOs. Stream: Statements (Occurrences).
* Kinds. Stream: Mappings (Roles).
* Quad Contexts. Stream: Transforms (Interactions).
* Functions:
* Applicable
* Statement: Statement (D) SPOs
* Statement: Mapping (C) Kinds
* Statement: Transform (I) Quad Contexts.
* Transforms Order:
* (C, CPrevStatement, CMapping, CNextStatement);