* Monad Wrapper Role Types Hierarchy:
* Transform : Mapping : Statement : Resource;
* URN : Resource (alignments). Primitives.
* Resource : Root Category. URN : Source / Surrogate Key / Crafted. Naming / Encodings (below).
* Statement : Resource. Wraps Resource as Statement Category.
* Mapping : Statement. Wraps Statement as Functional Transform Declaration Category.
* Transform : Mapping. Wraps Mapping as Functional Transform Application Category.
* Monads Wrapped Roles types / values. Transforms / Mappings. Objects.
* Resource Roles: Reified CSPO Resources types / values, Kinds types / values, Statements, Class, Instance, Occurrence, Occurring, Attribute, Context, Value, Role, Models Roles type / values getters (populated Mappings / Transforms.
* Resources Objects hierarchies / APIs: (Reified Mappings / Transforms): DTOs / Dynamic Functional, DTOs (hashmap) of CSPO roles getters / model domain browsing getters (Functional Transform parameterized Mappings). Flow context: referrers / keys types / values: address::city::street Aggregation / Map. Templates / Mappings: reified declarative Augmentation Data Flows.
* Ontology alignments: Data / Schema / Behavior Augmentations. Model / Schema / Upper / Domains: purposes / gestures (MVC / DCI Mappings / Transforms) layers. Example:
* Transforms: unaEmpresa::unEmpleado::unaPosicion::salary; Salary inferred by context and unaPosicion, unEmpleado, unaEmpresa.
* Class / Instance / Reified (occurrence / ocurring) / Mappings / Transforms:
* Statement
* Kind
* ContextKind (SubjectKind / ObjectKind flow signature)
* Statement
* SubjectKind
* PredicateKind
* ObjectKind
* Subject
* Predicate
* Object
* Resource
* Context
* Template
* Class
* Instance
* Occurrence
* Value
* Occurring / Context (Statements / Kinds)
* Sets
* Graphs
* Roles (Metaclass, Class, Occurrence, Context, Role)
* Dimensional (Dimension, Measure, Unit, Value)
* Discrete (Relationship, Relation, Kind, Entity)
* 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. OSGi 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). 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).
* 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.
* 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.
* (...)
* Layers, input layer: Model Roles. Aggregation: Layer Roles shifting until full Layers Roles Statements. Layer: Augments Models.
* Core Statements Roles Resource Interleaved Model (Infer Types, PKs, FKs):
* Occurrence: (Class, Instance, Attribute, Value);
* Occurring: (Class, Instance, Occurrence,  Role);
* Statement Transforms / Relations: order, equivalence, roles, etc.
* Resource of Resource Monad: Occurrence  / Occurrings Quads CSPO Members Aggregation Transforms.
* Resource(t : T) :: contexts :: subjects :: predicates :: objects : Resource(u : U)
* Resource of URNs: Aggregated CSPOs Transforms of Occurrences / Occurrings. Matchings.
* [Resource.of](http://resource.of)(Resource / Class, Instance, Occurring Attribute / Occurring URN, Attribute Value / Occurring Role) :: contexts :: subjects :: predicates :: objects : Resources(CSPO / Resource / occurrence / occurring : URN).
* Resources of Resources: Occurrences / Occurring CSPOs Transforms Matching Wrapped / Wrappings / Transforms Resource Types Shapes Matching Templates. Data flow, apply transforms, order, lists.
* [Resource.of](http://resource.of)(templ : Templ) :: contexts : Resources(c : Context).
* Layer Template Mappings:
* Template : Context : Statement : Resource Layer Roles Monads.
* Layers Quads Aggregation: rotating value role types from previous layer to next layer from V to C):
* Canonical Template Mapping Layers: Aggregation of Template Matching Models Layers until first layer Value is wrapped into final layer Context. Perform Layers Augmentations.
* Monads. Wraps Models Roles. Matching: Patterns wrapped Resource Roles values: placeholders, variables, wildcards, reified instances. Example: Template wraps previous layer Value as Context, Layer Context Value wrapped as Subject value (shifting).
* Core Roles (Wrappers):
* Resource : Monad(x : Resource);
* Statement : Monad(x : Resource) : Resource, Quad DTO.
* Context : Monad(x : Resource) : Statement;
* Template : Monad(x : Resource) : Context.
* Template : Context : Statement : Resource
* Input Layers: Core Statements Roles Resource Shapes: (interleaving graphs models) : Statements CSPO.
* Occurrence Input: (Class, Instance, Attribute, Value);
* Occurring Input: (Class, Instance, Occurrence, Role);
* Output Layer: (Template, Context, Statement, Resource); Input OPSC Resources wrapped in output Layer Roles.
* Sets, Graph, Roles: Services / Augmentations helper Models / Facades.
* Example:
* (Amantes, Pedro, Ama, María);
* (MariaAmadaPor, Amantes, Pedro, Ama);
* (Amar, MariaAmadaPor, Amantes, Pedro);
* (PedroAmaA, Amar, MariaAmadaPor, Amantes);
* (Amor, PedroAmaA, Amar, MariaAmadaPor);
* Augmentations:
* Model Augmentations:
* Alignment: Data Matching. Resources.
* Alignment::match
* Alignment::perform
* Alignment::greaterThan
* Alignment::equals
* Alignment::lessThan
* Activation: Schema Matching. Kinds.
* Activation::match
* Activation::perform
* Activation::superTypeOf
* Activation::sameTypeOf
* Activation::subTypeOf
* Aggregation: Behavior Matching. Contexts Flows.
* Aggregation::match
* Aggregation::perform
* Aggregation::beforeThan
* Aggregation::contains
* Aggregation::containedIn
* Aggregation::afterThan
* Domain Augmentations:
* Transforms Reified in Layers Contexts. Pattern Matching Template Layer resolved:
* Mapping::match
* Mapping::apply
* Mapping::Context
* Mapping::Subject
* Mapping::Predicate
* Mapping::Object
* Encode reified Template Mappings / Transforms. Patterns:
* Model / Domain Augmentations Mappings / Transforms:
* (Wrapper, Wrapped, Mapping, Transform);
* Next Layer step: match / apply Augmentations.
* Layers Template: Layer of CSPO Data Flow Patterns Resolution Resources : Reified (meta) Resources.
* Once Models Layers Matrix are built and populated / aggregated:
* Layers steps Augmentations: perform shifting and wrapping of aggregation values. Layer step Template Mapping: Layer::nextLayer : Layer, for each Layer, match / performs.
* Template, Mapping, Pattern
* (Template, Context, Statement, Resource);
* Resource :: occurrences :: roles :: contexts : Resource