* ObjectMapper Functional Model.
* AugmentationService (Registry, Naming, Index) instances shared Model State.
* Aggregation, Alignment, Activation: Syncronize Augmentation State.
* Registry, Index, Naming en functional-model: getURI, getStatement, getURIOccurrence (CSPOs Statement, URI, Kind URIs hash keys), getKinds (CSPOs) by URIs Registry de-serialization helper methods (shallow serialization, only URIs strings). Shared state across services invocations.
* Registry::getStatement populate CSPO Kinds with Registry::getCSPOKind. Default Kind: CSPO URIOccurrence URI.
* Occurrences: Serialization writeObject.
* URI Statements / CSPO Kinds Occurrences?
* Kind Occurrences Kind URI Occurrences?
* De-serialization: Populate Occurrences. Preserve PrimeIDs.
* Kind: URIOccurrence Attributes / Attributes, Values. CSPO Specialization.
* Initial Kinds: CSPO URIOccurrences URIs Kinds. Kind URI: Hash From Occurrence URI, Attributes Primes IDs (Update in Statements instantiation).
* Kinds Merge: Same Attributes, Attributes Subset / Superset (Prime IDs). Super Kind / Sub Kind. Update URIOccurrences with merged Kinds. Kind URI: Attributes Prime IDs.
* Wrapped Kind Constructor: CSPOs Kinds in other CSPOs Kinds Roles (i.e.: ValueKind from SubjectKind).
* Kind Statements URIOccurrences Kinds: Kinds URIs equals URIOccurrences URIs? CSPO Kinds extends CSPOs? CSPOs Interfaces? Kinds Interfaces?
* Meta Kind (Values): Attributes Values Prime IDs Product.
* Map de URIs / PrimeIDs en Registry / PrimesIDService. getID(URI) / getURI(Long ID). Aggregation ID Products. URIOccurrences per CSPOs.
* Augmentation: Aggregation, Alignement, Activation consumes input Statement streams (events), process (Functions: Statements URIOccurrences & URI IDs populated, calculate Kind IDs, links, matching, contexts), returns / publish (Augmentation caller subscribes) Augmented (Kind IDs populated) / Inferred (Alignment) / Context (Kinds Activation) Statements (Augmentation caller updates Registry, Naming, Index, Model State Context: Functions w./ Statement, Occurrence, Kind strategies).
* Controller / Services endpoints:
* Datasources publish triples. Augmentation listens, Statements URIs, Occurrences Factory / Model Services (Registry, Naming, Index).
* Flux<Statement> augment(Flux<Statement> stream)
* Flux<Statement> aggregate(Flux<Statement> stream) : Kind Augmented Statements;
* Flux<Statement> align(Flux<Statement> stream) : Aligned (links, matching) Statements;
* Flux<Statement> activate(Flux<Statement> stream) : Kind (Context / Relationship Statements (browseable instance Statements);
* Consumer Templates (DCI) from Activation Contexts / Relationships (Kind Statements). Browse / CRUD Instances Interactions. Consumes Augmentation outcomes (state, model datasources updated incrementally, Registry: Data, Naming: Context, Index: Interactions), Produces Quads (HATEOAS Datasource).
* ConsumerService: Spring HATEOAS.
* Statements / Kind Statements (Contexts / Relationships: Alignment / Activation).
* URIsFunction, URIOccurrencesFunction, KindsFunction, StatementsFunction. Strategy (Kind / Statement).
* Services posts messages, process streams (functions), invokes Services Controllers (callback response) Interactions.
* Augmentation (messages /streams / functions):
* Datasource : Plain (schema encoding) URI Strings Triples Input Streams. Consumes Augmentation Triples (sync).
* Registry: Factory. Aggregation Interactions. Model Types. URIOccurrences CSPO message stream, CSPO Kind response.
* Naming: Alignment Interactions. Model Link Prediction, Ontology Matching.
* Index: Activation Interactions. Model Templates.
* Consumer: HATEOAS.
* Monad<URI>;
* Function<URI, URI>(URI strategy);
* liftM2(URI, URI) : URI. Function<URI, URI, URI>(URI strategy).
* Functions: CRUD, Assertions, Predicates, Query, Browse, Augmentations: Aggregation, Alignment, Activation over Backend Graph Model Services (Naming, Index, Registry) streams.
* CUD / Assertions / Augmentations Functions results: update / sync Backend Graph Model Services (Naming, Index, Registry) before subscribe().
* Example: ApplyCSPOKind(Kind) : CSPO Kind, URI, Statements (Existing: Attributes / Values, Assignment: Attributes Placeholders). Get / Set CSPOs (downstream).
* Kinds occurrences recursive Kinds: hierarchy / meta Kind. Kind Occurrence role. Until object role (all attributes).
* Core Model: CSPO, Statements, Kinds. Functional Abstractions, etc. Projects Dependency. Rx Persistence Service / Repository (Augmentation Naming, Index, Registry Graph Management)
* Architecture / Components (Services):
* DatasourceService / Designer. Model
* AugmentationService (Naming, Index, Registry Graph Management) / Designer. Core Model
* AggregationService / Designer
* AlignmentService / Designer
* ActivationService / Designer
* ConsumerService / Designer (Templates: ETL). Model
* Rx Service APIs (COST Protocol) Events / Model Verbs (CRUD) Mappings
* Management Service APIs (REST Designers) Events / Model Verbs (CRUD) Mappings
* Statement : URI? CSPOs Prime IDs URI hash.
* Monad<URI>;
* Function<URI, URI>(URI strategy);
* liftM2(URI, URI) : URI. Function<URI, URI, URI>(URI strategy).
* Functions: CRUD, Assertions, Predicates, Query, Browse, Augmentations: Aggregation, Alignment, Activation over Backend Graph Model Services (Naming, Index, Registry) streams.
* CUD / Assertions / Augmentations Functions results: update / sync Backend Graph Model Services (Naming, Index, Registry) before subscribe().
* Example: ApplyCSPOKind(Kind) : CSPO Kind, URI, Statements (Existing: Attributes / Values, Assignment: Attributes Placeholders). Get / Set CSPOs (downstream).
* Kinds occurrences recursive Kinds: hierarchy / meta Kind. Kind Occurrence role. Until object role (all attributes).
* Architecture / Components (Services):
* DatasourceService / Designer
* AugmentationService (Naming, Index, Registry Graph Management) / Designer
  + AggregationService / Designer
  + AlignmentService / Designer
  + ActivationService / Designer
* ConsumerService / Designer (Templates: ETL)
* Rx Services APIs (COST Protocol) Events
* Management Service APIs (REST Designers) Events
* URIs como sets de Statements (ocurrencias):
* URI: Todos los Statements en que ocurre como S/P/O. Data Statements.
* KindURI: URI; Statements con Kind URI (en SPO correspondiente), Atributos, Valores (Meta Kinds): Definición. Context Statements: Kinds Roles (Ocurrencias).
* KindSPO : URI; Interaction Statements: Roles / Actors (Data) SPOs.
* Mono<URI>
* Flux<Statement>
* Function<URI, Set<Statement>>(arg : URI);
* Occurrences
* Definitions / Filters / Predicates
* URI / Kind arg : Interaction Statements;
* URI / KindSPO arg : Context Statements;
* Kind / URI arg : Interaction Statements;
* Kind / KindSPO arg : Data Statements;
* KindSPO / URI arg : Context Statements (Kind Definitions);
* KindSPO / Kind arg : Data Statements;
* Function<URI, URI>(arg : Set<Statement>); Functional CUD
* C, S, P, O Function<Statement, URI>
* Overloaded URI.apply(URI) : URI;
* Graph REST (Stateful, Dialog, Referrer Context / Interaction) Protocol. Query / Browse Contexts. Monads Activation (JAF). Addressable Interactions (state log, backend synchronization, DIDs).
* SPOs: Monads. Kinds: Functions. SPO Statements: Data, Kind Statements: Contexts. Kind / SPO Statements: Interactions.
* SPOs: URI Occurrences (in Statements). URIs knows its Occurrences. URIs Prime Factor ID. Statement ID: SPOs URIs Prime Factor IDs Product.
* Kinds: SPOs Occurrences. Statements Instance, Attributes, Values (SPOs). Kinds knows its Occurrences (Instances and Kind Statements Contexts). Kinds URIs: Kind Instances (SPOs) Occurrences URIs Prime Factor IDs Product.
* Kinds / SPOs Statement URIs: Kind Instances URIs Prime Factor IDs Product + / + SPO Occurrence URI Prime ID. SPOs knows its Kinds in Occurrences.
* SPOs URIs: [https://serviceHost/#URIPrimeCountID](https://servicehost/#URIPrimeCountID)
* Kinds URIs: [https://serviceHost/SubjectKind/#InstancesIDsPrimesProduct](https://servicehost/SubjectKind/#InstancesIDsPrimesProduct)
* Kinds / SPOs URIs: [https://serviceHost/SubjectKind/#InstancesIDsPrimesProduct/URIPrimeCountID](https://servicehost/SubjectKind/#InstancesIDsPrimesProduct/URIPrimeCountID)
* Registry Service: SPO URIs / Original URIs Mappings. Provenance sync. Aggregation. Data (SPO Statements).
* Naming Service: Kind Names from NLP. Alignment. Ontology Matching (sameAs). Contexts (Kind Statements).
* Index Service: Representations Message Resolution. Activation. Interactions (Kinds / SPOs Statements).
* Representations. Message Format: URI Occurrences / Statements browsing (DCI state building). REST CRUD: Services Endpoints. Statements Browse / Forms.
* Aggregation:
* Input: S, P, O.
* Output: Kind:S, Kind:P, Kind:O (in SPO occurrences)
* Kind: SPOs with SPOs Kinds / SPO Attributes / Values, SPO Occurrences (encoded / matching in SPO Kinds).
* Alignment:
* Common Attributes between Kinds occurring in linking Statements (S1, Attr1, O1; O1, Attr2, O2; S1, Attr2, O2). Paired Attributes by Kind.
* Example: Project / Language; Developer / Project; Developer / Language.
* Attributes paths attribute closures: S, brotherOf, O; O, fatherOf, O2; S unkleOf O2.
* Alignment triples (inferred):
* unkleKind, brotherKind, fatherKind; nephewKind, fatherKind, brotherKind;
* grandSonKind, fatherKind, fatherKind; grandFatherKind, sonKind, sonKind;
* yernoKind, spouseKind, parentKind; suegroKind, sonKind, spouseKind.
* cousinKind, unkleKind, sonKind;
* Activation:
* CSPO Contexts (Kind Roles / SPO Actors). Apply Context / SPO Kinds according Context Kinds / Roles (Attributes / Values). Order.
* Output: Kind:C, Kind:S, Kind:P, Kind:O (in CSPO Context occurrences) SPOs.
* Order / Occurrences: Kinds Hierarchies / Statements IDs.
* Aggregation: Instance Type (Class: Attributes Features), Instance Kind (Meta Type: Class Attributes / Attributes Values Features). Classification Labels (Encoding). Kind: Type / Meta type.
* Alignment: Specialization. Parent Type Attributes Features / Parent Kind Attributes / Values Specialized Instance Type / Kind (Attributes / Values) Regression Placeholders.
* Activation: Developer / Project, Project / Language, Developer / Language. Context Roles Clustering Links Prediction. Predicates: Roles. Subject: Actors, Objects: Data.
* Topics.
* ML. LLMs. GenAI.
* Graphs. GQL.
* Rx. Microservices.
* Functional.
* Downloads. Online resources. Links.
* Documents.
* Graphs Book.
* Tools.
* Spring.
* RDF4J / Jena.
* Neo4j.
* Spark. EDA.
* NLP. LLMs.
* Scrapbook.
* Augmentation. Feedback. Serialization layers formats / schemes to / from models transformations (Functional / XSLT):
* Aggregation (Type inference): Data (Model Typed Instances / Attributes / Values).
* Alignment (Relationship inference): Context (Model Typed Placeholders).
* Activation (Role inference): Interaction (Model Sequence Statements).
* Dimensions. Placeholders. Declaratively state computations / transformations shapes / results mappings (FPGA / ALU). Lookups. COST (referrer).
* Translation: Context(Role, LHS, Role, RHS); (Context, LHS, Role, RHS);
* Context, Role: LLM Classification (topics). NER. Concepts networks (graphs, (isA: Type, Instance, Attribute: hasA, Value); sameAs.
* Semiotics: Syntax (signs: encoding: values / types) data, Grammar (signs: relationships, roles) context / information, Semantics (signs: objects, de-reference in context) interactions / knowledge.
* MLM: Infer Placeholders.
* NSP: Infer Data / Context / Interactions sequences.
* Feature extraction.
* Naming (Kinds): LLMs.
* Index (Topics: Roles, Data, Contexts, etc.): LLMs.
* Registry (Hierarchical key / value: TMRM / LHS, RHS): LLMs.
* Codat (metamodel encoded layers):
* Prompt for schema / model and ask for data (Data)
* Prompt for data and ask for code (Contexts)
* Prompt for code and ask for execution (Interaction)
* Data Science:
* Statistics.
* Python.
* EDA.
* SQL.
* ML.
* Models.
* Big Data (Hadoop / Spark)
* Tensorflow workspace.
* Ollama workspace.
* Ollama / WebUI
* Dockerfile Ollama
* Streamlit
* Langchain (SQL, RAG).
* LlamaIndex.
* Fine tuning models.
* JSON to CSV Prompt.
* Dictionary. NLP.
* Graph Alignment.
* Naming, Index (masks / patterns completion), Registry (key / value, graphs, TMDM / TMRM): LLMs.
* Alignment:
* Ontology Matching, Merge, Augmentation.
* Aggregation:
* Types (Attribuytes). Kinds (Attributes, Values). Roles (Relationships). Contexts (Instances).
* Activation:
* Functional Kinds Monad. Context State: Dimension Kind Value, Monad application axis, Type Kind wq, me Dimension occurrences. Person(Date, Employer). Mutations: Feedback.
* Relationship: (Type, Instance, Attribute, Value);
* (Kind, SPO, SPO, SPO);
* (SK, S, P, O);
* (PK, P, S, O);
* (OK, O, P, S);
* Contexts: Types: Class; Kinds: Roles; Data: Attributes / Values.
* Execution: Crawl (state, functional entities). Direction / Feedback: referrer, runat headers. Request / response addressable (DIDs) state aggregated, aligned, activated entities.
* Addressable state transitions (Functional, DIDs): Alignment, Augmentation, Activation Execution statements results.
* Write. Relationships (available input prompts):
* (Kind, Entity, KindAttributesDomain, AttributesRangeValue);
* Read. Relationships (available browseable rels):
* (Kind, InstanceResultHolder, AttributePredicateFilter, AttributePredicateValue);
* Order: Available (assert) / Browseable (Cons lists): Translation / Composition: Distance, Speed, Time. Order: before, during, after. Containment. Relation properties.
* Dimension: Comparable.
* Dimension: (Unit, Measure / Instance, Prev Dimension, Next Dimension);
* Equivalence: (Unit, Measure / Instance, Equiv Dimension, Equiv Dimension);
* OrderStmt: (Dimension, Instance, OrderStmt, OrderStmt);
* (Speed, anSpeed, Distance, Time);
* (Speed, anSpeed, Time, Distance);
* (Distance, aDistance, Speed, Time);
* (Distance, aDistance, Time, Speed);
* (Time, aTime, Distance, Speed);
* (Time, aTime, Speed, Distance);
* (Containment, aContainment, Container, Containee);
* (Container, aContainer, Containment, Containee);
* (Containee, aContainee, Containment, Container);
* Tensorflow
* Big Data
* Rx
* Data Science
* MML Book
* NLP / Embeddings / Knowledge Graph
* ML
* Graphs ML
* FCA / Graphs
* Rx
* DDD (DCI, DOM)
* Microservices
* Spring Boot REST HAL Browser
* <https://causeway.apache.org/>
* Spark
* RDF / OWL / SPARQL
* GNN Ontology Matching / Alignment / Augmentation (link prediction)
* KIE
* Aggregation, Alignment (matching / rules) / Augmentation.
* Feedback: Generative Modelling.
* Embeddings: numerical representations.
* Parser: Graph.
* Spark.
* Example: S (Atoms), P (Bonds), O (Atoms). Kind: substance type.
* Feedbacks siempre a Aggregation Inputs. Order: sequences Comparators.
* Aggregation: Types. Association rule mining. Queries / Views: FCA / ML (Transforms: Features / Outputs).
* Input: Graphs (Provenance, S, P, O). Adapters (Provenance I/O): RDBMS, XML, JSON, CSV.
* Outputs: (Kind, Instance, Attribute, Value)
* Feedback Outputs: Aggregation Inputs.
* Alignment: Link prediction. Matching. DOM (Reactive: Rules / Relationships / Order).
* Inputs: (Kind, Instance, Attribute, Value)
* Outputs: (Rule, Kind, Attribute, Value)
* Outputs: (Kind, Instance, Attribute, Kind)
* Feedback Outputs: Aggregation Inputs.
* Activation: Roles in contexts interactions. DDD / DCI (Functional Kinds)
* Inputs: Alignment Outputs.
* Outputs: (Context, Interaction, Kind, Instance)
* Feedback Outputs: Alignment Inputs.
* Instance, Attribute, Value: CSPOs.
* Component Model:
* Reactive Components API: Functions Library (Layers): Function Kind. Rules.
* Predict Feedback Outputs.
* Custom Keras Layers Models (ML Pipeline).
* Services: Custom Layers Models (ML Backend). Streams Endpoints.
* Feedbacks: ML Models.
* Docs: Google Docs, Readme, Code. Topics (lectures).
* Association rule mining. Regression.
* BiFunction: consumes / produces.
* Aggregate entities / attributes / values by entity / attribute / value types (by Kinds). Type facet / role in occurrence / relation Context: (Context, SubjectKind INST, PropertyKind ATTR, ModelObjectKind VAL).
* Kind instance: all Kind's (Employee) Statements. Aggregated Kind instance: all Kind's instances (anEmployee) Statements.
* Grammar Rules (Context, Concept): Transforms (map) input Kind instance (Statements) attributes / values to output Kind Statements (attributes / values).
* Alignment, Augmentation, Matching Rules: Consumes / Produces Statements. Intermediate mappings: CSPO, Kinds. Infer mappings between Kinds (common INST, ATTR, VAL).
* Rules: Comparator<CONSUMES, PRODUCES> (Contexts / Concepts). Kinds example: (Single : Kind, Marriage : Rule Concept, Married : Kind); (Married : Kind, Divorce : Rule Concept, Divorced : Kind); (Father : Kind, Brother : Rule Concept, Uncle : Kind); Concept: Rule mapping of Attributes (previous: CONSUMES, next: PRODUCES). Order output Statements / CSPOs / Kinds / Resources (List): map(rule::orderOutput).
* Context Consumes / Produces: Father Kind, Son Kind. Apply sets Son Instance uncle Instance attribute from Concept mapping. Uncle Object Kind.
* Rule.compose(context); andThen(concept);
* Context produces Rule consumes. Consumes Rule consumes. Mapping function according Context Rule.
* Concept consumes Rule produces. Produces Rule produces. Mapping function according Concept Rule.
* Resource scheme (Statements, CSPOs, Kinds). Registry: labels, metadata (i.e. Concepts). RDFS / OWL Aligned. HasResource (Statement, CSPO ResourceOccurrence, Kinds).
* Rule.matches (CONSUMES, PRODUCES): types, terminals (Resource), primitives (resolve).
* Rules: apply. Statement(CSPOs(Kind(Resource))). Parent Context: Child Concept. Nested apply(CONSUMES) : PRODUCES.
* CONSUMES / PRODUCES: Mono.just / apply Concept Rule. Statements: Flux stream / apply Rules.
* Rules: ResourceOccurrence lhs / rhs. Recursion contexts / concepts till primitive rules (sameAs, map, match. See RDFS/OWL relationships. Monad transforms expects lhs, emits rhs), lhs / rhs terminals resolution / Resource matching / ops. Statements Resources. LHS / RHS: Monads (Statements, Kinds, CSPOs, Resources).
* Flux from Statements apply Rules to stream, subscribe emits Aligned, Augmented, Matched Statements / CSPOs / Kinds / Resources.
* Kinds: OWL Classes, restriction on allValuesFrom Attributes.
* Statements, CSPOs, Resource. ResourceOccurrence.
* OntClass is a facet of Resource.
* Marshall Results: RDF / RDFS / OWL. Parsing results: Alignment, Augmentation, Matching. Rules (Inference, Aggregation, ML, FCA, SPARQL). Materialize Rule Statements (inference). Activation: Aligned DOM / DCI RDF4J DAOs.
* CSPO Resource / ResourceOccurrences y Kind Resource ResourceOccurrences (CSPOs).
* Textual Representation: Parser / XSLT Templates emitting same Textual input (echo rules parsing). Parsing: DTOs AST.
* DOM / DCI Alignments: Activation. RDF4J DAOs.
* Kinds: a rdfs:type.
* CSPOs rdfs:type Kind.
* Labels. S, P. O / Literal no tiene?
* Implications: confidence, support. Attibutes: Association rule mining. Values: inference / regression / inter CSPO Contexts Statements (placeholders). Add inferred Statements.
* Spring HAL / REST HATEOAS.
* Augmented Statements Representation: JSON, XML, HAL, RDF Model. Parser I/O Augmented Statements. Parser: parsecj / XML XSLT.
* Grammar, Rules / Productions: Augmentation. Parsing (parsecj).
* Grammar, Rules / Productions: Alignment (upper). Parsing (parsecj).
* Grammar, Rules / Productions: Matching (merge). Parsing (parsecj).
* Parsing: Non Terminals, Terminals, Production Rules.
* Non Terminals:
* Statement :: (Context, Subject, Predicate, Object);
* Subject :: (Kind, Statement, Resource, Kind);
* Kind :: (Resource, INST, ATTR, VAL);
* Resource :: (IRI, CSPO, Statement, Kind);
* IRI :: string;
* Terminals / Inputs: Aggregation Type Augmented Statements Stream.
* Operations / Relationships Productions (stated / inferred):
* (Context, LHS, Concept, RHS);
* Apply LHS to Context, Concept to prev result, RHS to prev result.
* Augmentation Rules. State order (Concept event):
* (Context, Statement, Statement, Statement);
* Alignment Rules (upper onto). State order (Concept event):
* (Context, Kind, Kind, Kind);
* Matching Rules:
* (Context, Resource, Resource, Resource);
* Spring Rest Repositories HAL Explorer
* FCA Implication confidence / support.
* Resource: getResourceOccurences. CSPOs / Kinds in context Statements.
* Rules: (semiotics context / concept) Grammar. Statements: Productions: Alignment Statements (ResourceOccurrences Rule Kinds). Terminals (ResourceOccurrence) / non Terminals (Kinds) / Primitives / Relationships (Context Concepts): state Matching Rules.
* (Context ContextKind, LHS ResourceOccurrence, Context Concept Kind / Context ResourceOccurrence Kind, RHS ResourceOccurrence);
* Apply Context function Kind to input ResourceOccurrences Flux : LHS. Apply Context Concept function Kind to LHS Flux : RHS.
* (Amor, (Pedro, amaA, Maria), amada, María);
* (Amor, (Pedro, amaA, Maria), Maria, amada);
* (Amor, Pedro, (Pedro, amaA, Maria), amante);
* (Amor, amante, Pedro, (Pedro, amaA, Maria));
* (Amor, Pedro, amante, (Pedro, amaA, Maria));
* (Amor, Maria, Pedro, amada);
* (Empleo, Maria, Pedro, compañera);
* (Familia, padre, hermano, tío);
* (Familia, hijo, (padre, hermano, tío), tío);
* (Empleo, (Pedro, trabajaPara, unEmpleador), (unEmpleador, brindaServiciosPara, unCliente), (Pedro, brindaServiciosPara, unCliente));
* (etc.: CSPO, Kinds, Statements). ResourceOccurrences LHS, Concepts (ResourceOccurrence Context Kind), RHS:
* (Statement, CSPO, Kind)
* (Statement, Kind, CSPO)
* (Statement, Statement, Statement)
* (Kind, Statement, CSPO)
* (Kind, CSPO, Statement)
* (Kind, Kind, Kind)
* (CSPO, Statement, Kind)
* (CSPO, Kind, Statement)
* (CSPO, CSPO, Kind)
* Rules Aggregation: Rule Context application matches / filters input Statements Flux for LHS Statements, Kinds, CSPOs Flux. Concepts Aggregated by Context CSPOs, Kinds, Statements. RHS result of applying Concept Kind to LHS Flux (infer Grammar).
* StatementKind : ContextKind;
* Ontology Matching: state (inferred) equivalence between types, instances, attributes, relationships and values. Rules / Grammar.
* Ontology Merge / Align: state (inferred) equivalence Statements and Upper Ontology mappings between types, instances, attributes, relationships and values. Rules/ Grammar.
* ContextKind: Aggregate Property Attributes (Employment), Object Values (Employeer). Context Resource: Subject Kind.
* Ontology Matching: Rules / Implications. Kinds instances, attributes, values. Order.
* Rules, Implications Statements: ordered Lists (subsumption / LHS domain, RHS range: extract SubjectKind from ObjectKind by inverse relationship). Comparator against Statements product: lt (sub / next), gt (super / prev), eq (aligned / matched).
* Rule Statement (Kinds) ResourceOccurrences: Implication Statements (Kinds CSPOs). Statements Resources: reified bnode, RDF\*.
* Apply Kind (Function) to Monad wrapped ResourceOccurrences. Statements: CSPO Kinds. CSPOs: Kinds occurrences. Kinds: Relationship (Concept).
* Contexts / ContextKinds: Implications / Rules (Upper asserted / Aligned Knowledge / Primitives). (Context, LHS, Concept, RHS);
* Implication / Assertion Statement: (Context, Subject, Property, ModelObject);
* ContextKind Aggregation. Instance: Context, Atribute: Concept, Value: RHS, from Assertion Statements.
* Rule Statement: (Context: ContextKind, LHS: SubjectKind, Concept: PropertyKind. RHS: ObjectKind.
* Aggregate Kinds into Rule Statements. (KindStatements). Statements match Contexts, match SK, apply Concept, match OK.
* Materialize Rule Statements from Model into CSPOs Occurrences from Resources:
* Rule: (Amor, Amante, Ama, Amada);
* Statements: (unAmor, pedro, amaA, maría);
* Rule: (Son, Father, BrotherOf, Uncle);
* Statements: (aSon, aSonFather, brotherOfFather, aSonUncle);
* Rules Alignment: RuleConcepts / RuleImplications. Aggregated from CSPO Occurrences, Kind Relationships, Statements / KindStatements Relationships. Rule / Implication: (Context, LHS, Concept, RHS); Populate Rules with Upper Alignment knowledge and aligned Model inferred knowledge.
* Rule / Implication: (Context, LHS, Concept, RHS) Inference: Kinds Instance, Attribute, Value CSPO Kinds Contexts aggregation / subsumption (domain / range flow). Matching: CSPO Kind Attributes matching those of Rules / Concepts.
* Relationship Statements: Rules (LHS, RHS in a Context by a Concept Property) stated via Upper Aligned types, instances, attributes, values Statements, Kinds, CSPOs and Resources (Concepts / Implications).
* Relationships / Rules Model. Model Aggregation / Inference.
* Implication (instances): Statements, Kinds, Resources. Parse instances as new Rules / Concepts.
* Model (Upper Aligned CSPOs):
* Rule / Implication: (Context, LHS, Concept, RHS);
* LHS, RHS : ResourceOccurrence (Statement / Kind / CSPO) wrapper Monad.
* Context, Concept : Kind. Rule / Implication. Rules reified by Kinds: Rule / RuleKind (Aggregated RuleKindStatements): (ContextKind, InstanceLHS, ConceptAttributeKind, ValueRHS). Stated / Inferred Rules / Implications RuleKindStatements.
* RuleKind: ContextKind; Instance: LHS ResourceOccurrence; Attribute: (Property) Kind, Value: LHS ResourceOccurrence. (wrapped ResourceOccurrences). RuleKinds / RuleKindStatements match.
* CSPO Contexts: Rules / Implications. ContextKinds: Contexts / Concepts. Initial Rules: input Statements. Aggregate RuleKinds (Aligned / Upper Populated RuleKindStatements).
* Kinds : Function::apply(Kind arg) : Kind (resolve instances, attributes, values occurrences).
* ResourceOccurrence wrapping Monad (Resource, Context, Kind), Kind Functions (domain / range, subsumption).
* Resolve Rule Context Kind Resources. Feed Rule LHS Resources.
* Apply Rule Concept Kind to previous Rule LHS Kind Resources output.
* Map Rule Concept Kind outputs into RHS ResourceOccurrence result.
* (Context, LHS, Concept, RHS); Upper / Aligned Primitives: (LHS, RHS). Primitive Contexts, Concepts.
* Rule Aggregation: Each Statement is itself a Rule stating a single fact building a Concept by means of its Kinds relationships / CSPO inter Statements occurrences.
* Initial Rules / Implications: initial Statements, KindStatements. Aggregate / Align Statements CSPO Kinds / KindStatements Rules / Implications by their attribute / value CSPO types Kinds.
* Rule: (Kind) Statement / Kind / Resource matching LHS (Upper onto matching aligned), (Upper / Model inferred) Concept in Context, (Kind) Statement, Kind, Resource RHS.
* State: Type according property values / property values relationships. Alignment, Augmentation: shapes (Upper OWL, SHACL, ShEx). Metakinds (person / adult: age value gt 21; brother / uncle: brother :hasChild). Order relationships (Statements). Activation (Context inter-resource relationships). Reification: express inter (Kind) Statements relationships. Statements ResourceOccurrence (Kinds / CSPOs) of Concepts Resources. Relationship Statements (upper / shapes).
* Upper Ontology / Matching: State implies Event / Relationship / Type: maritalStatus, uncle. Employee isA Worker. State facts by means of Upper types, instances, properties, values, relationships (RelationshipStatements: Statements about Statements, Resources, Kinds concepts).
* RelationshipStatements (Kinds / CSPO, Relationships (reified Statements). Aggregate inferred rules / logic statements (Resolvers / Notation3). Upper ontology types, instances, properties, values, relationships (sameAs).
* Upper Ontology Relationships: property values according type. before, during, after. Greater than, less than, equal. Event. Cause / Effect (state flow). Contains (Relationship, Place, Duration). Relationship kinds. N-ary Relationships (roles, actors templates).
* Augmentation / Alignment: Attribute / Links prediction. Upper Ontology alignment. Ontology Matching. Predicted Attributes Values.
* Activation: Kinds order (in Relationship Contexts).
* Order Statements by Kind hierarchies, previous / next common Attributes values.
* Label (add to Resource) Resolvers. RDFS#label NLP NER.
* Upper Ontology: ISO Topic Maps (TMDM, TMRM), ISO 15926 (OWL Templates).
* Statements / Kinds: order. Domain / range dataflows. Alignment, link prediction (between Kinds, KindStatements FCA).
* Activation: Kinds Functions (order: domain / range). Apply (map) adds (transition fits by schema) / query for Attributes / Values to Resource Kind Monad occurrence.
* Activation: Order Statements / Kinds transitions. Single, Married. Married, Divorced. Temporal order (octal comparison values encoding). Previous(Current / Next), Current(Previous / Next), Next(Current / Previous) Kinds.
* Inferred Attributes: Value Regression / Prediction (occurrences model). Scaling, MultiValuedContext.
* Inferred Attributes: Values prompts / placeholders (Resolvers).
* CSPO Kinds: Parameterized Instances, Attributes, Values.
* Kinds Impl: get inst (instances, instance), attrs(inst), vals(inst, attr) CSPOs.
* Kinds get Instance, Attribute, Value CSPOs. Fix / Move to CSPO Kinds.
* Input Statements: (Class, Instance, Attribute, Value). From RDF4J Model / Query.
* Query Repository rdfs:type value: Context, SPO.
* Kind Labels: from string similarity Subjects / Predicates NER.
* Reduce Context (FCA).
* Kinds hierarchies: Concepts subsumption.
* Activation: SubjectKinds / ObjectKinds: DCI Roles. PropertyKinds: Roles Behavior (Kinds Functions). KindStatements: DCI Contexts. Statements: DCI Interactions. CSPOs: Actor Occurrences.
* Alignment / Activation: Materialize Statements from KindStatements Kinds instances, attributes, values. FCA Objects / Attributes: Kinds.
* Activation: Rest (JAF), Restful Objects, HAL, Spring REST. HATEOAS.
* Alignment: Set KindStatement Kinds Kind.
* Aggregation: returns KindStatements. Kinds.getCSPOs.
* Scaled Attributes: Values.
* Kinds / Statements occurrences Resources, Kinds (roles), Statements (RDF star, reification).
* CSPO Statements.
* KindStatements
* RelationshipStatements (Kinds / CSPO, Relationships (reified Statements).
* KindStatement: schema Statements.
* Kinds Statements implements Kinds interfaces, Statement extends ResourceOccurrenceImpl (Context Statement Kinds: roles / metaclass: domain / range). CSPO Statements: Kind Statement Instances (Kind Statement getStatements).
* Reification Statements: example: SubjectKind implements Subject. Subjects SubjectKinds SubjectKindStatements.
* Kind Statements (Aggregated): SubjectKindStatement extends Statement. C: Super Kind, S: Kind, P: PK (domain), O: OK (range), From merged Occurrence Context Statement Kinds. Kind Statement Kinds: role / metaclass.
* CSPOs: Resource IRI / PrimeID. Merged Kinds: Instance, Attribute, Value Context Lattice products.
* Resource: Function and Monad wrapped value. Domain / range state / flow context / interactions order. Relationships. Activation.
* Statements URN: lookUp by CSPO hashCode (Map / Embedding).
* Statement IRI: Reification, RDF-Star (URN Encoded).
* Resource PrimeID, Resources URN Encoding.
* Kinds Aggregation: Merged Resource IRI (Instance, Attribute, Value URN). Prime factors ID. FCA4J.
* FCA Contexts:
* getContextAggregatedKinds
* getSubjectAggregatedKinds
* getPropertyAggregatedKinds
* getObjectAggregatedKinds
* FCA. Primes Embeddings. Property contexts, Subject objects, Objects attributes. Aggregate by Property contexts. Lattice Graph (Alignment).
* Serialize Aggregation, Alignment, Activation into RDF / RDFS / OWL / JSON-LD: Instantiate Model via RDF4J APIs. (Align upper ontologies / [schema.org](http://schema.org)).
* JSON-LD Serialization (ObjectMapper). Types documents: Kinds, Resources. AggregationService performAggregation.
* Convert any input into triples graph (Statements). Alignment: Graph Deep Learning). DOM Mappings. UIMA / Tika / any23 / D2RQ.
* Kinds: Initial Kind Resource: Kind Instance Resource. Aggregate: Instances Context (Attrs / Values).
* Target: Generate typed RDF RDFS / OWL (Kinds). Sets Aggregation.
* Target: Discover Links / Relationships. DOM Augmentation.
* Target: Discover Data Contexts / Interactions. DCI Activation.
* Kinds : Statement (CK Kind IRi)
* Kinds interface / impls: get / set Inst, Attr, Val (Statement ctx).
* getSubjectAggregatedKind()
* Resource / ResourceImpl occurrences / hashCode.
* CSPOs, Kinds, Statements: IRIs Occurrences roles. IRI in CSPOs, CSPOs in Kinds / Statements, Kinds in Statements. Resource get occurrences (Subjects, etc. streams / filter). Resource super type (IRI / Occurrences) CSPO / Kinds / Statement parameterized (Statement Resource<CSPOs, Kinds>).
* Statements: Model / Model Statements wrapper. Streams (load statements, aggregate kinds, populate CSPOs, filter / occurrences: CSPO / Kinds Statements). Statements (filter / occurrences set streams, Statement instances).
* Todos los atributos de un Subject son un Kind: CK (Kind Statements Stream). Kinds (filter / occurrences sets streams, Kind instances).
* Merge Kind Statements. Kind IRI: Statement Contexts.
* Populate CSPOs. Filter Statements Stream. Subjects, etc. (filter / occurrences set streams, Subject, etc. instances).
* Quads (Resource) : Type, Instance, Attribute, Value (parameterized). Interfaces.
* Resource : IRI, Occurrences (CSPOs, Kinds, Statements). Implementations.
* Context
* Subject (SK, IRI, Ps, Os)
* Property
* Object
* SubjectKind (Super SK, SK, Ps, Os) : P / O
* PropertyKind : S / O
* ObjectKind : P / S
* ContextKind : (S, P, O)
* Statement (CSPO, CK:SK:PK:OK Combinations, SK en CK: Ps / Os).
* Functions (Transforms) : Reactive.
* Getters: Type, Instances, Attributes, Values
* Getters CSPOs, Kinds, Statements Occurrences
* Alignment: super Kind infer sub Kind (superset) Attributes (Aggregation) / Values (Regression).
* Aggregation: Kinds (IRIs) / Instances (Statement Occurrences) Attributes / Values. Sets Model. Classification (no labels)
* Alignment: Types / Instances Relationships / Links. Kinds / Attributes. DOM Model (DAOs).
* Activation: Roles / State in Contexts / Interactions Relationships. DCI Model. Clustering.
* FCA Embeddings.
* Statements / Kind Statements : IRI (toString / hashCode). Reification.
* D2RQ Sample Dataset (Cinema).
* Aggregation: Materialize Kind Statements (RDFS / OWL).
* Debug Aggregation results.
* Retrieveable IRIs : JSON-LD. Alignment: [schema.org](http://schema.org).
* Alignment: sameAs.
* Context Kind: Context, Predicate, Object.
* Kind (role) in Occurrences / Context Kind Statements. DCI Context, DOM Alignment.
* SK: Subject, Predicate, Object.
* Debug: Aggregation. Sample relational dataset (D2RQ) FKs (inferred / Alignment). Browsable Aggregation output format: Statements, Materialized Kind Statements: RDFS / OWL (Semantic Web / SPARQL Browser).
* Reactive Functional / Async:
* Flat map each 3 aggregations from initial flows, each return next aggregation input. Flat map 3 aggregation groups.
* FKs. Objects / Subjects referred by IRI, Aggregated end types (Kinds / Alignment), Relationship Kind (PredicateKind). Star Schema (Dimensions, Facts).
* IRIStatementOccurrence / Kind getStatementOccurrences.
* Alignment: query / views / rules parameters: add OWL Templates to Repository Model. DOM DAOs.
* Materialize inferred Statements (reified Kind Statements) in Repository temporal Model. Alignment: SPARQL query templates / views / rules. RDF4JTemplate: DOM DAOs. Services.
* Materialize RDF rdf:class (Kinds), rdf:type, etc. Inferred attributes. Alignment queries.
* Materialize: Statements: Reification / RDF\*. Statement : rdfs:Statement. IRIStatementOccurrence. Statement / Kind Statement IRI.
* Handle Literals: Literals, Map<hashCode, Literal>. IRI: Literal type + hashCode.
* toString: Complete Entities / FKs JSON. Messaging: parse Resources toString. Autowired Custom ObjectMapper.
* Convert Aggregation to Functional / Reactive. Integrate with RDF4J.
* Aggregation. Learn Types.
* Alignment. Learn Relationships.
* Activation. Learn Interactions.
* Alignment: Functional / Reactive APIs. DOM / DAOs. Embeddings.
* Alignment Service: align(), get Types, Entities, Attributes, Values. RDF4JTemplate DAOs.
* Resource (HasIRI) interface. Resource::apply(Resource res) : Resources; Dynamic Functions.
* Map / flatMap Model domain (Static Functions).
* Zip T1, T2 (Resources product): apply T2 to T1 (Dynamic Functions) : Resources.
* Alignment: Events, Order relationship. Inferred.
* Update RDF4J with Alignment inferred Relationships (query / rules SPARQL).
* Emit inferred Statements.
* TODO:
* Kind has Attributes.
* Attributes has Kind according Occurrence (Instance).
* Attribute has Value according Instance.
* Values has Attribute according Occurrence (Kind).
* Replace placeholder IRIs with primes product URNs (Embeddings).
* Kind, Instance, Attribute, Value: IRIs / Embeddings.
* Primes.common(BigInteger, BigInteger).
* Primes.contains(BigInteger, BigInteger).
* Kinds Naming: Context IRI. KindStatements. Contexts Label.
* Aggregation: order. Kinds sub / super Kinds relationship encoded in Statement Context IRI.
* Kind reification: S : (ParentSK, SK, P, O);
* Alignment: Links Attributes / Values inference / materialization.
* Alignment: Theory. Order. State. Rules. Events. Relations. Uncle : (aFather, fatherBrother)
* Alignment: Attribute / Value clustering determines meta-Kind (age > 21 : Person age Adult Adult Person); (birthPlace BsAs, Person nationality Argentina, Argentinian Person).
* Alignment: Upper Ontology. Mappings.
* Alignment: Ontology Matching.
* Activation (DDD DOM / DCI). REST Dialog front end.
* Activation Service: activate(), get Interactions, Roles, Actors. RDF4JTemplate DAOs.
* Embeddings calculation (FCA / primes. Contexts / Attributes / Objects / Concepts.
* FCA Contexts: Axis, Objects, Attributes, Concepts.
* Restful DOM / OGM (DCI, CDI)
* Activation: Gestures (available Transforms).
* HATEOAS REST (HAL) Architecture. Components:
* Alignment / Activation DOM (RDF4JTemplate DAOs):
* Entity
* Name
* Type
* Value
* atributes : Set<Entity>
* Type : Entity
* Name
* players : Set<Entity>
* members : Set<Type>
* Role : Type (Wrapper)
* Type
* Declarative Behavior / Relations?
* Context : Role (Composite)
* Entity
* Type
* Role
* Declarative Behavior / Relations?
* Interaction
* bindings : Set<Context>
* Model:
* Statement : Resource
* Context : Context
* Subject : Subject
* Predicate : Predicate
* Object : Object
* Kind : Statement
* Kind<INST super Resource, ATTR super Resource, VAL super Resource>
* Context : Kind<> (this)
* Subject : INST
* Predicate : ATTR
* Object : VAL
* List<Kind<>> : statements
* Context : Resource (Wrapper)
* Statement : context
* ContextKind : role
* player : Resource
* ContextKind : Context, Kind
* extends Kind<Context, Subject, Predicate>
* Subject : Resource (Wrapper)
* Statement : context
* SubjectKind : role
* player : Resource
* SubjectKind : Subject, Kind
* extends Kind<Subject, Predicate, Object>
* Predicate : Resource (Wrapper)
* Statement : context
* PredicateKind : role
* player : Resource
* PredicateKind : Predicate, Kind
* extends Kind<Predicate, Subject, Object>
* Object : Resource (Wrapper)
* Statement : context
* ObjectKind : role
* player : Resource
* ObjectKind : Object, Kind
* extends Kind<Object, Subject, Predicate>
* Alignment:
* Ontology Matching, Merge, Augmentation.
* Aggregation:
* Types (Attributes). Kinds (Attributes, Values). Roles (Relationships). Contexts (Instances).
* Activation:
* Functional Kinds Monad. Context State: Dimension Kind Value, Monad application axis, Type Kind Dimension occurrences. Person(Date, Employer). Mutations: Feedback.
* Relationship: (Type, Instance, Attribute, Value);
* (Kind, SPO, SPO, SPO);
* (SK, S, P, O);
* (PK, P, S, O);
* (OK, O, P, S);
* Contexts: Types: Class; Kinds: Roles; Data: Attributes / Values.
* Execution: Crawl (state, functional entities). Direction / Feedback: referrer, runat headers. Request / response addressable (DIDs) state aggregated, aligned, activated entities.
* Addressable state transitions (Functional, DIDs): Alignment, Augmentation, Activation Execution statements results.
* Write. Relationships (available input prompts):
* (Kind, Entity, KindAttributesDomain, AttributesRangeValue);
* Read. Relationships (available browseable rels):
* (Kind, InstanceResultHolder, AttributePredicateFilter, AttributePredicateValue);
* Order: Available (assert) / Browseable (Cons lists): Translation / Composition: Distance, Speed, Time. Order: before, during, after. Containment. Relation properties.
* Dimension: Comparable.
* Dimension: (Unit, Measure / Instance, Prev Dimension, Next Dimension);
* Equivalence: (Unit, Measure / Instance, Equiv Dimension, Equiv Dimension);
* OrderStmt: (Dimension, Instance, OrderStmt, OrderStmt);
* (Speed, anSpeed, Distance, Time);
* (Speed, anSpeed, Time, Distance);
* (Distance, aDistance, Speed, Time);
* (Distance, aDistance, Time, Speed);
* (Time, aTime, Distance, Speed);
* (Time, aTime, Speed, Distance);
* (Containment, aContainment, Container, Containee);
* (Container, aContainer, Containment, Containee);
* (Containee, aContainee, Containment, Container);
* Alignment: Ontology Matching, Merge, Augmentation.
* Aggregation: Types (Attributes). Kinds (Attributes, Values). Roles (Relationships). Contexts (Instances).
* Activation: Functional Kinds Monad. Context State: Dimension Kind Value, Monad application axis, Type Kind Dimension occurrences. Person(Date, Employer). Mutations: Feedback.
* Feedbacks siempre a Aggregation Inputs. Order: sequences Comparators.
* Aggregation: Types. Association rule mining. Queries / Views: FCA / ML (Transforms: Features / Outputs).
* Input: Graphs (Provenance, S, P, O). Adapters (Provenance I/O): RDBMS, XML, JSON, CSV.
* Outputs: (Kind, Instance, Attribute, Value)
* Feedback Outputs: Aggregation Inputs.
* Alignment: Link prediction. Matching. DOM (Reactive: Rules / Relationships / Order).
* Inputs: (Kind, Instance, Attribute, Value)
* Outputs: (Rule, Kind, Attribute, Value)
* Outputs: (Kind, Instance, Attribute, Kind)
* Feedback Outputs: Aggregation Inputs.
* Activation: Roles in contexts interactions. DDD / DCI (Functional Kinds)
* Inputs: Alignment Outputs.
* Outputs: (Context, Interaction, Kind, Instance)
* Feedback Outputs: Alignment Inputs.
* Instance, Attribute, Value: CSPOs.
* Component Model:
* Reactive Components API: Functions Library (Layers): Function Kind. Rules.
* Predict Feedback Outputs.
* Custom Keras Layers Models (ML Pipeline).
* Services: Custom Layers Models (ML Backend). Streams Endpoints.
* Feedbacks: ML Models.
* Si te ponés a pensar el paradigma de grafos abarca todos los formatos de representación y almacenamiento. En una DB relacional vos podés exportar a un grafo (Sujeto, Atributo, Valor) lo que en la DB está como (PK, Columna, ValorCelda). Igualmente para CSV, JSON, XML, etc.
* Entonces, en respecto a sus entradas y salidas, para una arquitectura Big Data se podrían delinear las siguientes capas:
* 1) Aggregation: Inferencia de tipos de los componentes de los triples de un grafo. Según sus atributos, un sujeto pertenece al tipo de los sujetos que comparten sus atributos. FCA.
* Inputs: Triples Grafo (Sujeto, Atributo, Objeto).
* Outputs: (Tipo, Instancia, Atributo, Valor).
* Feedback: Inputs de capa Aggregation.
* 2) Alignment: Inferencia de atributos y valores para un sujeto (association rule / link prediction). Matching de componentes de grafos equivalentes (ontology matching). DOM.
* Inputs: (Tipo, Instancia, Atributo, Valor)
* Outputs: (Tipo, Instancia, Atributo, Tipo)
* Feedback: Aggregation Inputs.
* 3) Activation: Inferencia de roles en interacciones instancias de contextos. DCI / DDD.
* Inputs: (Tipo, Instancia, Atributo, Tipo)
* Outputs: (Contexto, Interacción, Tipo, Instancia)
* Feedback: Alignment Inputs.
* Hasta acá es una jerarquía de capas mejor implementada con servicios / functional programming. El tema son los 'Feedbacks':
* El Feedback de cada capa podría ser resultado de invocar un modelo ML entrenado en una arquitectura específica de entradas y salidas de cada capa para predecir que datos de entrada generados hubiesen causado determinada salida.
* De este modo las capas aprenden de las entradas y salidas entradas que hubiesen generado resultados posibles.
* Un ejemplo sería el input: (Pedro, Empleador, IBM) en Aggregation, cuyo output sería: (Empleado, Pedro, Empleador, IBM) y que tendría el feedback: (IBM, EmpleadorDe, Pedro) con el output: (Empleador, IBM, Empleado, Pedro).
* Con la suficiente metadata (antiguedad, posición, seniority, etc) se podría llegar, por regression, al feedback: (Pedro, Salario, SalarioDePedro : valor / placeholder / tipo dentro de un rango).
* Referencias:
* FCA: https://en.m.wikipedia.org/wiki/Formal\_concept\_analysis
* DOM: https://www.geeksforgeeks.org/dynamic-modelling-in-object-oriented-analysis-and-design
* DCI: https://en.m.wikipedia.org/wiki/Data,\_context\_and\_interaction
* DDD: https://en.m.wikipedia.org/wiki/Domain-driven\_design
* Association rule mining. Regression.
* BiFunction: consumes / produces.
* Aggregate entities / attributes / values by entity / attribute / value types (by Kinds). Type facet / role in occurrence / relation Context: (Context, SubjectKind INST, PropertyKind ATTR, ModelObjectKind VAL).
* Kind instance: all Kind's (Employee) Statements. Aggregated Kind instance: all Kind's instances (anEmployee) Statements.
* Grammar Rules (Context, Concept): Transforms (map) input Kind instance (Statements) attributes / values to output Kind Statements (attributes / values).
* Alignment, Augmentation, Matching Rules: Consumes / Produces Statements. Intermediate mappings: CSPO, Kinds. Infer mappings between Kinds (common INST, ATTR, VAL).
* Rules: Comparator<CONSUMES, PRODUCES> (Contexts / Concepts). Kinds example: (Single : Kind, Marriage : Rule Concept, Married : Kind); (Married : Kind, Divorce : Rule Concept, Divorced : Kind); (Father : Kind, Brother : Rule Concept, Uncle : Kind); Concept: Rule mapping of Attributes (previous: CONSUMES, next: PRODUCES). Order output Statements / CSPOs / Kinds / Resources (List): map(rule::orderOutput).
* Context Consumes / Produces: Father Kind, Son Kind. Apply sets Son Instance uncle Instance attribute from Concept mapping. Uncle Object Kind.
* Rule.compose(context); andThen(concept);
* Context produces Rule consumes. Consumes Rule consumes. Mapping function according Context Rule.
* Concept consumes Rule produces. Produces Rule produces. Mapping function according Concept Rule.
* Resource scheme (Statements, CSPOs, Kinds). Registry: labels, metadata (i.e. Concepts). RDFS / OWL Aligned. HasResource (Statement, CSPO ResourceOccurrence, Kinds).
* Rule.matches (CONSUMES, PRODUCES): types, terminals (Resource), primitives (resolve).
* Rules: apply. Statement(CSPOs(Kind(Resource))). Parent Context: Child Concept. Nested apply(CONSUMES) : PRODUCES.
* CONSUMES / PRODUCES: Mono.just / apply Concept Rule. Statements: Flux stream / apply Rules.
* Rules: ResourceOccurrence lhs / rhs. Recursion contexts / concepts till primitive rules (sameAs, map, match. See RDFS/OWL relationships. Monad transforms expects lhs, emits rhs), lhs / rhs terminals resolution / Resource matching / ops. Statements Resources. LHS / RHS: Monads (Statements, Kinds, CSPOs, Resources).
* Flux from Statements apply Rules to stream, subscribe emits Aligned, Augmented, Matched Statements / CSPOs / Kinds / Resources.
* Kinds: OWL Classes, restriction on allValuesFrom Attributes.
* Statements, CSPOs, Resource. ResourceOccurrence.
* OntClass is a facet of Resource.
* Marshall Results: RDF / RDFS / OWL. Parsing results: Alignment, Augmentation, Matching. Rules (Inference, Aggregation, ML, FCA, SPARQL). Materialize Rule Statements (inference). Activation: Aligned DOM / DCI RDF4J DAOs.
* CSPO Resource / ResourceOccurrences y Kind Resource ResourceOccurrences (CSPOs).
* Textual Representation: Parser / XSLT Templates emitting same Textual input (echo rules parsing). Parsing: DTOs AST.
* DOM / DCI Alignments: Activation. RDF4J DAOs.
* Kinds: a rdfs:type.
* CSPOs rdfs:type Kind.
* Labels. S, P. O / Literal no tiene?
* Implications: confidence, support. Attibutes: Association rule mining. Values: inference / regression / inter CSPO Contexts Statements (placeholders). Add inferred Statements.
* Spring HAL / REST HATEOAS.
* Augmented Statements Representation: JSON, XML, HAL, RDF Model. Parser I/O Augmented Statements. Parser: parsecj / XML XSLT.
* Grammar, Rules / Productions: Augmentation. Parsing (parsecj).
* Grammar, Rules / Productions: Alignment (upper). Parsing (parsecj).
* Grammar, Rules / Productions: Matching (merge). Parsing (parsecj).
* Parsing: Non Terminals, Terminals, Production Rules.
* Non Terminals:
* Statement :: (Context, Subject, Predicate, Object);
* Subject :: (Kind, Statement, Resource, Kind);
* Kind :: (Resource, INST, ATTR, VAL);
* Resource :: (IRI, CSPO, Statement, Kind);
* IRI :: string;
* Terminals / Inputs: Aggregation Type Augmented Statements Stream.
* Operations / Relationships Productions (stated / inferred):
* (Context, LHS, Concept, RHS);
* Apply LHS to Context, Concept to prev result, RHS to prev result.
* Augmentation Rules. State order (Concept event):
* (Context, Statement, Statement, Statement);
* Alignment Rules (upper onto). State order (Concept event):
* (Context, Kind, Kind, Kind);
* Matching Rules:
* (Context, Resource, Resource, Resource);
* Spring Rest Repositories HAL Explorer
* FCA Implication confidence / support.
* Resource: getResourceOccurences. CSPOs / Kinds in context Statements.
* Rules: (semiotics context / concept) Grammar. Statements: Productions: Alignment Statements (ResourceOccurrences Rule Kinds). Terminals (ResourceOccurrence) / non Terminals (Kinds) / Primitives / Relationships (Context Concepts): state Matching Rules.
* (Context ContextKind, LHS ResourceOccurrence, Context Concept Kind / Context ResourceOccurrence Kind, RHS ResourceOccurrence);
* Apply Context function Kind to input ResourceOccurrences Flux : LHS. Apply Context Concept function Kind to LHS Flux : RHS.
* (Amor, (Pedro, amaA, Maria), amada, María);
* (Amor, (Pedro, amaA, Maria), Maria, amada);
* (Amor, Pedro, (Pedro, amaA, Maria), amante);
* (Amor, amante, Pedro, (Pedro, amaA, Maria));
* (Amor, Pedro, amante, (Pedro, amaA, Maria));
* (Amor, Maria, Pedro, amada);
* (Empleo, Maria, Pedro, compañera);
* (Familia, padre, hermano, tío);
* (Familia, hijo, (padre, hermano, tío), tío);
* (Empleo, (Pedro, trabajaPara, unEmpleador), (unEmpleador, brindaServiciosPara, unCliente), (Pedro, brindaServiciosPara, unCliente));
* (etc.: CSPO, Kinds, Statements). ResourceOccurrences LHS, Concepts (ResourceOccurrence Context Kind), RHS:
* (Statement, CSPO, Kind)
* (Statement, Kind, CSPO)
* (Statement, Statement, Statement)
* (Kind, Statement, CSPO)
* (Kind, CSPO, Statement)
* (Kind, Kind, Kind)
* (CSPO, Statement, Kind)
* (CSPO, Kind, Statement)
* (CSPO, CSPO, Kind)
* Rules Aggregation: Rule Context application matches / filters input Statements Flux for LHS Statements, Kinds, CSPOs Flux. Concepts Aggregated by Context CSPOs, Kinds, Statements. RHS result of applying Concept Kind to LHS Flux (infer Grammar).
* StatementKind : ContextKind;
* Ontology Matching: state (inferred) equivalence between types, instances, attributes, relationships and values. Rules / Grammar.
* Ontology Merge / Align: state (inferred) equivalence Statements and Upper Ontology mappings between types, instances, attributes, relationships and values. Rules/ Grammar.
* ContextKind: Aggregate Property Attributes (Employment), Object Values (Employeer). Context Resource: Subject Kind.
* Ontology Matching: Rules / Implications. Kinds instances, attributes, values. Order.
* Rules, Implications Statements: ordered Lists (subsumption / LHS domain, RHS range: extract SubjectKind from ObjectKind by inverse relationship). Comparator against Statements product: lt (sub / next), gt (super / prev), eq (aligned / matched).
* Rule Statement (Kinds) ResourceOccurrences: Implication Statements (Kinds CSPOs). Statements Resources: reified bnode, RDF\*.
* Apply Kind (Function) to Monad wrapped ResourceOccurrences. Statements: CSPO Kinds. CSPOs: Kinds occurrences. Kinds: Relationship (Concept).
* Contexts / ContextKinds: Implications / Rules (Upper asserted / Aligned Knowledge / Primitives). (Context, LHS, Concept, RHS);
* Implication / Assertion Statement: (Context, Subject, Property, ModelObject);
* ContextKind Aggregation. Instance: Context, Atribute: Concept, Value: RHS, from Assertion Statements.
* Rule Statement: (Context: ContextKind, LHS: SubjectKind, Concept: PropertyKind. RHS: ObjectKind.
* Aggregate Kinds into Rule Statements. (KindStatements). Statements match Contexts, match SK, apply Concept, match OK.
* Materialize Rule Statements from Model into CSPOs Occurrences from Resources:
* Rule: (Amor, Amante, Ama, Amada);
* Statements: (unAmor, pedro, amaA, maría);
* Rule: (Son, Father, BrotherOf, Uncle);
* Statements: (aSon, aSonFather, brotherOfFather, aSonUncle);
* Rules Alignment: RuleConcepts / RuleImplications. Aggregated from CSPO Occurrences, Kind Relationships, Statements / KindStatements Relationships. Rule / Implication: (Context, LHS, Concept, RHS); Populate Rules with Upper Alignment knowledge and aligned Model inferred knowledge.
* Rule / Implication: (Context, LHS, Concept, RHS) Inference: Kinds Instance, Attribute, Value CSPO Kinds Contexts aggregation / subsumption (domain / range flow). Matching: CSPO Kind Attributes matching those of Rules / Concepts.
* Relationship Statements: Rules (LHS, RHS in a Context by a Concept Property) stated via Upper Aligned types, instances, attributes, values Statements, Kinds, CSPOs and Resources (Concepts / Implications).
* Relationships / Rules Model. Model Aggregation / Inference.
* Implication (instances): Statements, Kinds, Resources. Parse instances as new Rules / Concepts.
* Model (Upper Aligned CSPOs):
* Rule / Implication: (Context, LHS, Concept, RHS);
* LHS, RHS : ResourceOccurrence (Statement / Kind / CSPO) wrapper Monad.
* Context, Concept : Kind. Rule / Implication. Rules reified by Kinds: Rule / RuleKind (Aggregated RuleKindStatements): (ContextKind, InstanceLHS, ConceptAttributeKind, ValueRHS). Stated / Inferred Rules / Implications RuleKindStatements.
* RuleKind: ContextKind; Instance: LHS ResourceOccurrence; Attribute: (Property) Kind, Value: LHS ResourceOccurrence. (wrapped ResourceOccurrences). RuleKinds / RuleKindStatements match.
* CSPO Contexts: Rules / Implications. ContextKinds: Contexts / Concepts. Initial Rules: input Statements. Aggregate RuleKinds (Aligned / Upper Populated RuleKindStatements).
* Kinds : Function::apply(Kind arg) : Kind (resolve instances, attributes, values occurrences).
* ResourceOccurrence wrapping Monad (Resource, Context, Kind), Kind Functions (domain / range, subsumption).
* Resolve Rule Context Kind Resources. Feed Rule LHS Resources.
* Apply Rule Concept Kind to previous Rule LHS Kind Resources output.
* Map Rule Concept Kind outputs into RHS ResourceOccurrence result.
* (Context, LHS, Concept, RHS); Upper / Aligned Primitives: (LHS, RHS). Primitive Contexts, Concepts.
* Rule Aggregation: Each Statement is itself a Rule stating a single fact building a Concept by means of its Kinds relationships / CSPO inter Statements occurrences.
* Initial Rules / Implications: initial Statements, KindStatements. Aggregate / Align Statements CSPO Kinds / KindStatements Rules / Implications by their attribute / value CSPO types Kinds.
* Rule: (Kind) Statement / Kind / Resource matching LHS (Upper onto matching aligned), (Upper / Model inferred) Concept in Context, (Kind) Statement, Kind, Resource RHS.
* State: Type according property values / property values relationships. Alignment, Augmentation: shapes (Upper OWL, SHACL, ShEx). Metakinds (person / adult: age value gt 21; brother / uncle: brother :hasChild). Order relationships (Statements). Activation (Context inter-resource relationships). Reification: express inter (Kind) Statements relationships. Statements ResourceOccurrence (Kinds / CSPOs) of Concepts Resources. Relationship Statements (upper / shapes).
* Upper Ontology / Matching: State implies Event / Relationship / Type: maritalStatus, uncle. Employee isA Worker. State facts by means of Upper types, instances, properties, values, relationships (RelationshipStatements: Statements about Statements, Resources, Kinds concepts).
* RelationshipStatements (Kinds / CSPO, Relationships (reified Statements). Aggregate inferred rules / logic statements (Resolvers / Notation3). Upper ontology types, instances, properties, values, relationships (sameAs).
* Upper Ontology Relationships: property values according type. before, during, after. Greater than, less than, equal. Event. Cause / Effect (state flow). Contains (Relationship, Place, Duration). Relationship kinds. N-ary Relationships (roles, actors templates).
* Augmentation / Alignment: Attribute / Links prediction. Upper Ontology alignment. Ontology Matching. Predicted Attributes Values.
* Activation: Kinds order (in Relationship Contexts).
* Order Statements by Kind hierarchies, previous / next common Attributes values.
* Label (add to Resource) Resolvers. RDFS#label NLP NER.
* Upper Ontology: ISO Topic Maps (TMDM, TMRM), ISO 15926 (OWL Templates).
* Statements / Kinds: order. Domain / range dataflows. Alignment, link prediction (between Kinds, KindStatements FCA).
* Activation: Kinds Functions (order: domain / range). Apply (map) adds (transition fits by schema) / query for Attributes / Values to Resource Kind Monad occurrence.
* Activation: Order Statements / Kinds transitions. Single, Married. Married, Divorced. Temporal order (octal comparison values encoding). Previous(Current / Next), Current(Previous / Next), Next(Current / Previous) Kinds.
* Inferred Attributes: Value Regression / Prediction (occurrences model). Scaling, MultiValuedContext.
* Inferred Attributes: Values prompts / placeholders (Resolvers).
* CSPO Kinds: Parameterized Instances, Attributes, Values.
* Kinds Impl: get inst (instances, instance), attrs(inst), vals(inst, attr) CSPOs.
* Kinds get Instance, Attribute, Value CSPOs. Fix / Move to CSPO Kinds.
* Input Statements: (Class, Instance, Attribute, Value). From RDF4J Model / Query.
* Query Repository rdfs:type value: Context, SPO.
* Kind Labels: from string similarity Subjects / Predicates NER.
* Reduce Context (FCA).
* Kinds hierarchies: Concepts subsumption.
* Activation: SubjectKinds / ObjectKinds: DCI Roles. PropertyKinds: Roles Behavior (Kinds Functions). KindStatements: DCI Contexts. Statements: DCI Interactions. CSPOs: Actor Occurrences.
* Alignment / Activation: Materialize Statements from KindStatements Kinds instances, attributes, values. FCA Objects / Attributes: Kinds.
* Activation: Rest (JAF), Restful Objects, HAL, Spring REST. HATEOAS.
* Alignment: Set KindStatement Kinds Kind.
* Aggregation: returns KindStatements. Kinds.getCSPOs.
* Scaled Attributes: Values.
* Kinds / Statements occurrences Resources, Kinds (roles), Statements (RDF star, reification).
* CSPO Statements.
* KindStatements
* RelationshipStatements (Kinds / CSPO, Relationships (reified Statements).
* KindStatement: schema Statements.
* Kinds Statements implements Kinds interfaces, Statement extends ResourceOccurrenceImpl (Context Statement Kinds: roles / metaclass: domain / range). CSPO Statements: Kind Statement Instances (Kind Statement getStatements).
* Reification Statements: example: SubjectKind implements Subject. Subjects SubjectKinds SubjectKindStatements.
* Kind Statements (Aggregated): SubjectKindStatement extends Statement. C: Super Kind, S: Kind, P: PK (domain), O: OK (range), From merged Occurrence Context Statement Kinds. Kind Statement Kinds: role / metaclass.
* CSPOs: Resource IRI / PrimeID. Merged Kinds: Instance, Attribute, Value Context Lattice products.
* Resource: Function and Monad wrapped value. Domain / range state / flow context / interactions order. Relationships. Activation.
* Statements URN: lookUp by CSPO hashCode (Map / Embedding).
* Statement IRI: Reification, RDF-Star (URN Encoded).
* Resource PrimeID, Resources URN Encoding.
* Kinds Aggregation: Merged Resource IRI (Instance, Attribute, Value URN). Prime factors ID. FCA4J.
* FCA Contexts:
* getContextAggregatedKinds
* getSubjectAggregatedKinds
* getPropertyAggregatedKinds
* getObjectAggregatedKinds
* FCA. Primes Embeddings. Property contexts, Subject objects, Objects attributes. Aggregate by Property contexts. Lattice Graph (Alignment).
* Serialize Aggregation, Alignment, Activation into RDF / RDFS / OWL / JSON-LD: Instantiate Model via RDF4J APIs. (Align upper ontologies / schema.org).
* JSON-LD Serialization (ObjectMapper). Types documents: Kinds, Resources. AggregationService performAggregation.
* Convert any input into triples graph (Statements). Alignment: Graph Deep Learning). DOM Mappings. UIMA / Tika / any23 / D2RQ.
* Kinds: Initial Kind Resource: Kind Instance Resource. Aggregate: Instances Context (Attrs / Values).
* Target: Generate typed RDF RDFS / OWL (Kinds). Sets Aggregation.
* Target: Discover Links / Relationships. DOM Augmentation.
* Target: Discover Data Contexts / Interactions. DCI Activation.
* Kinds : Statement (CK Kind IRi)
* Kinds interface / impls: get / set Inst, Attr, Val (Statement ctx).
* getSubjectAggregatedKind()
* Resource / ResourceImpl occurrences / hashCode.
* CSPOs, Kinds, Statements: IRIs Occurrences roles. IRI in CSPOs, CSPOs in Kinds / Statements, Kinds in Statements. Resource get occurrences (Subjects, etc. streams / filter). Resource super type (IRI / Occurrences) CSPO / Kinds / Statement parameterized (Statement Resource<CSPOs, Kinds>).
* Statements: Model / Model Statements wrapper. Streams (load statements, aggregate kinds, populate CSPOs, filter / occurrences: CSPO / Kinds Statements). Statements (filter / occurrences set streams, Statement instances).
* Todos los atributos de un Subject son un Kind: CK (Kind Statements Stream). Kinds (filter / occurrences sets streams, Kind instances).
* Merge Kind Statements. Kind IRI: Statement Contexts.
* Populate CSPOs. Filter Statements Stream. Subjects, etc. (filter / occurrences set streams, Subject, etc. instances).
* Quads (Resource) : Type, Instance, Attribute, Value (parameterized). Interfaces.
* Resource : IRI, Occurrences (CSPOs, Kinds, Statements). Implementations.
* Context
* Subject (SK, IRI, Ps, Os)
* Property
* Object
* SubjectKind (Super SK, SK, Ps, Os) : P / O
* PropertyKind : S / O
* ObjectKind : P / S
* ContextKind : (S, P, O)
* Statement (CSPO, CK:SK:PK:OK Combinations, SK en CK: Ps / Os).
* Functions (Transforms) : Reactive.
* Getters: Type, Instances, Attributes, Values
* Getters CSPOs, Kinds, Statements Occurrences
* Alignment: super Kind infer sub Kind (superset) Attributes (Aggregation) / Values (Regression).
* Aggregation: Kinds (IRIs) / Instances (Statement Occurrences) Attributes / Values. Sets Model. Classification (no labels)
* Alignment: Types / Instances Relationships / Links. Kinds / Attributes. DOM Model (DAOs).
* Activation: Roles / State in Contexts / Interactions Relationships. DCI Model. Clustering.
* FCA Embeddings.
* Statements / Kind Statements : IRI (toString / hashCode). Reification.
* D2RQ Sample Dataset (Cinema).
* Aggregation: Materialize Kind Statements (RDFS / OWL).
* Debug Aggregation results.
* Retrieveable IRIs : JSON-LD. Alignment: schema.org.
* Alignment: sameAs.
* Context Kind: Context, Predicate, Object.
* Kind (role) in Occurrences / Context Kind Statements. DCI Context, DOM Alignment.
* SK: Subject, Predicate, Object.
* Debug: Aggregation. Sample relational dataset (D2RQ) FKs (inferred / Alignment). Browsable Aggregation output format: Statements, Materialized Kind Statements: RDFS / OWL (Semantic Web / SPARQL Browser).
* Reactive Functional / Async:
* Flat map each 3 aggregations from initial flows, each return next aggregation input. Flat map 3 aggregation groups.
* FKs. Objects / Subjects referred by IRI, Aggregated end types (Kinds / Alignment), Relationship Kind (PredicateKind). Star Schema (Dimensions, Facts).
* IRIStatementOccurrence / Kind getStatementOccurrences.
* Alignment: query / views / rules parameters: add OWL Templates to Repository Model. DOM DAOs.
* Materialize inferred Statements (reified Kind Statements) in Repository temporal Model. Alignment: SPARQL query templates / views / rules. RDF4JTemplate: DOM DAOs. Services.
* Materialize RDF rdf:class (Kinds), rdf:type, etc. Inferred attributes. Alignment queries.
* Materialize: Statements: Reification / RDF\*. Statement : rdfs:Statement. IRIStatementOccurrence. Statement / Kind Statement IRI.
* Handle Literals: Literals, Map<hashCode, Literal>. IRI: Literal type + hashCode.
* toString: Complete Entities / FKs JSON. Messaging: parse Resources toString. Autowired Custom ObjectMapper.
* Convert Aggregation to Functional / Reactive. Integrate with RDF4J.
* Aggregation. Learn Types.
* Alignment. Learn Relationships.
* Activation. Learn Interactions.
* Alignment: Functional / Reactive APIs. DOM / DAOs. Embeddings.
* Alignment Service: align(), get Types, Entities, Attributes, Values. RDF4JTemplate DAOs.
* Resource (HasIRI) interface. Resource::apply(Resource res) : Resources; Dynamic Functions.
* Map / flatMap Model domain (Static Functions).
* Zip T1, T2 (Resources product): apply T2 to T1 (Dynamic Functions) : Resources.
* Alignment: Events, Order relationship. Inferred.
* Update RDF4J with Alignment inferred Relationships (query / rules SPARQL).
* Emit inferred Statements.
* TODO:
* Kind has Attributes.
* Attributes has Kind according Occurrence (Instance).
* Attribute has Value according Instance.
* Values has Attribute according Occurrence (Kind).
* Replace placeholder IRIs with primes product URNs (Embeddings).
* Kind, Instance, Attribute, Value: IRIs / Embeddings.
* Primes.common(BigInteger, BigInteger).
* Primes.contains(BigInteger, BigInteger).
* Kinds Naming: Context IRI. KindStatements. Contexts Label.
* Aggregation: order. Kinds sub / super Kinds relationship encoded in Statement Context IRI.
* Kind reification: S : (ParentSK, SK, P, O);
* Alignment: Links Attributes / Values inference / materialization.
* Alignment: Theory. Order. State. Rules. Events. Relations. Uncle : (aFather, fatherBrother)
* Alignment: Attribute / Value clustering determines meta-Kind (age > 21 : Person age Adult Adult Person); (birthPlace BsAs, Person nationality Argentina, Argentinian Person).
* Alignment: Upper Ontology. Mappings.
* Alignment: Ontology Matching.
* Activation (DDD DOM / DCI). REST Dialog front end.
* Activation Service: activate(), get Interactions, Roles, Actors. RDF4JTemplate DAOs.
* Embeddings calculation (FCA / primes. Contexts / Attributes / Objects / Concepts.
* FCA Contexts: Axis, Objects, Attributes, Concepts.
* Restful DOM / OGM (DCI, CDI)
* Activation: Gestures (available Transforms).
* HATEOAS REST (HAL) Architecture. Components:
* Alignment / Activation DOM (RDF4JTemplate DAOs):
* Entity
* Name
* Type
* Value
* atributes : Set<Entity>
* Type : Entity
* Name
* players : Set<Entity>
* members : Set<Type>
* Role : Type (Wrapper)
* Type
* Declarative Behavior / Relations?
* Context : Role (Composite)
* Entity
* Type
* Role
* Declarative Behavior / Relations?
* Interaction
* bindings : Set<Context>
* Model:
* Statement : Resource
* Context : Context
* Subject : Subject
* Predicate : Predicate
* Object : Object
* Kind : Statement
* Kind<INST super Resource, ATTR super Resource, VAL super Resource>
* Context : Kind<> (this)
* Subject : INST
* Predicate : ATTR
* Object : VAL
* List<Kind<>> : statements
* Context : Resource (Wrapper)
* Statement : context
* ContextKind : role
* player : Resource
* ContextKind : Context, Kind
* extends Kind<Context, Subject, Predicate>
* Subject : Resource (Wrapper)
* Statement : context
* SubjectKind : role
* player : Resource
* SubjectKind : Subject, Kind
* extends Kind<Subject, Predicate, Object>
* Predicate : Resource (Wrapper)
* Statement : context
* PredicateKind : role
* player : Resource
* PredicateKind : Predicate, Kind
* extends Kind<Predicate, Subject, Object>
* Object : Resource (Wrapper)
* Statement : context
* ObjectKind : role
* player : Resource
* ObjectKind : Object, Kind
* extends Kind<Object, Subject, Predicate
* Aggregate entities / attributes / values by entity / attribute / value types (by Kinds). Type facet / role in occurrence / relation Context: (Context, SubjectKind INST, PropertyKind ATTR, ModelObjectKind VAL).
* Kind instance: all Kind's (Employee) Statements. Aggregated Kind instance: all Kind's instances (anEmployee) Statements.
* Grammar Rules (Context, Concept): Transform (map) input Kind instance (Statements) attributes / values to output Kind Statements (attributes / values).
* Alignment, Augmentation, Matching Rules: Consumes / Produces Statements. Intermediate mappings: CSPO, Kinds. Infer mappings between Kinds (common INST, ATTR, VAL).
* Rules: Comparator<CONSUMES, PRODUCES> (Contexts / Concepts). Kinds example: (Single : Kind, Marriage : Rule Concept, Married : Kind); (Married : Kind, Divorce : Rule Concept, Divorced : Kind); (Father : Kind, Brother : Rule Concept, Uncle : Kind); Concept: Rule mapping of Attributes (previous: CONSUMES, next: PRODUCES). Order output Statements / CSPOs / Kinds / Resources (List): map(rule::orderOutput).
* Context Consumes / Produces: Father Kind, Son Kind. Apply sets Son Instance uncle Instance attribute from Concept mapping. Uncle Object Kind.
* Rule<CONSUMES, PRODUCES> : Function
  + Context : Rule
  + CONSUMES
  + Concept : Rule
  + PRODUCES
  + apply(CONSUMES) : PRODUCES
* Rules : Functional getters objects (Flux::map) "produces" augmented, aligned, matched "consumes".
* Rule.compose(context); andThen(concept);
* Context produces Rule consumes. Consumes Rule consumes. Mapping function according Context Rule.
* Concept consumes Rule produces. Produces Rule produces. Mapping function according Concept Rule.
* Resource scheme (Statements, CSPOs, Kinds). Registry: labels, metadata (i.e. Concepts). RDFS / OWL Aligned. HasResource (Statement, CSPO ResourceOccurrence, Kinds).
* Rules: ResourceOccurrence lhs / rhs. Recursion contexts / concepts till primitive rules (sameAs, map, match. See RDFS/OWL relationships. Monad transforms expects lhs, emits rhs), lhs / rhs terminals resolution / Resource matching / ops. Statements Resources. LHS / RHS: Monads (Statements, Kinds, CSPOs, Resources).
* Flux from Statements apply Rules to stream, subscribe emits Aligned, Augmented, Matched Statements / CSPOs / Kinds / Resources.
* Grammar, Rules / Productions: Augmentation. Parsing (parsecj).
* Grammar, Rules / Productions: Alignment (upper). Parsing (parsecj).
* Grammar, Rules / Productions: Matching (merge). Parsing (parsecj).
* Parsecj. XML / XSLT.
* Rule : (Context : Rule / Kind, LHS, Concept : Rule, RHS);
* Parsing: Non Terminals, Terminals, Production Rules.
* Non Terminals:
  + Statement :: (Context, Subject, Predicate, Object);
  + Subject :: (Kind, Statement, Resource, Kind);
  + Kind :: (Resource, INST, ATTR, VAL);
  + Resource :: (IRI, CSPO, Statement, Kind);
  + IRI :: string;
* Aggregation Model Textual Representation: Parse (Build Grammar / Rules: JParsec / XML, XSLT) to perform Augmentation, Alignment, Matching.
* Grammar: Augmentation, Alignment, Matching Rules. Model. API (Parsers API Matches Input). JParsec, XML / XSLT.
* Output: Parsed Textual Representation. RDF / RDFS / OWL Statements. Activation Model Aligned AST (DOM / DCI RDF4J Spring DAOs).
* <Statement>
  + <Context>
    - <Resource>
      * <IRI><IRI>
      * <ResourceOccurrences>
        + <ResourceOccurrence></ResourceOccurrence>
      * </ResourceOccurrences>
    - </Resource>
    - <ContextKind>
      * <Resource>
        + <IRI><IRI>
        + <ResourceOccurrences>

<ResourceOccurrence></ResourceOccurrence>

* + - * + </ResourceOccurrences>
      * </Resource>
      * <Instances>
        + <Instance>

<Context/>

<Attributes>

<Attribute>

<Property/>

<Values>

<Value>

<Object/>

</Value>

</Values>

</Attributes>

* + - * + </Instance>
      * </Instances>
    - </ContextKind>
  + </Context>
  + <Subject>
    - <Resource>
      * <IRI><IRI>
      * <ResourceOccurrences>
        + <ResourceOccurrence></ResourceOccurrence>
      * </ResourceOccurrences>
    - </Resource>
    - <SubjectKind>
      * <Resource>
        + <IRI><IRI>
        + <ResourceOccurrences>

<ResourceOccurrence>(...)</ResourceOccurrence>

* + - * + </ResourceOccurrences>
      * </Resource>
      * (Instances, Attributes, Values).
    - </SubjectKind>
  + </Subject>
  + <Property>
    - <Resource>
      * <IRI><IRI>
      * <ResourceOccurrences>
        + <ResourceOccurrence>(...)</ResourceOccurrence>
      * </ResourceOccurrences>
    - </Resource>
    - <PropertyKind>
      * <Resource>
        + <IRI><IRI>
        + <ResourceOccurrences>

<ResourceOccurrence>(...)</ResourceOccurrence>

* + - * + </ResourceOccurrences>
      * </Resource>
      * (Instances, Attributes, Values).
    - </PropertyKind>
  + </Property>
  + <Object>
    - <Resource>
      * <IRI><IRI>
      * <ResourceOccurrences>
        + <ResourceOccurrence>(...)</ResourceOccurrence>
      * </ResourceOccurrences>
    - </Resource>
    - <ObjectKind>
      * <Resource>
        + <IRI><IRI>
        + <ResourceOccurrences>

<ResourceOccurrence></ResourceOccurrence>

* + - * + </ResourceOccurrences>
      * </Resource>
      * (Instances, Attributes, Values).
    - </ObjectKind>
  + </Object>
* </Statement>
* Rules / Productions (stated / inferred Parsers / Templates):
  + (Context : Rule, Concepts template-match, LHS : Consumes (match), Concept : Rule (apply-templates), RHS : Produces (emits);
  + Context / Concept: Rule. Input Type: LHS / Output Type: RHS (domain / range Concept event ordering).
  + Apply LHS to Context, Concept to prev result, RHS to prev result.
* Rules that emit Rules / Productions Statements.
* Statements emitted by Rule application: Rule reference.
* Concepts: Mappings in Context between Rules domain / range. Access to occurring Context Rule instance.
* Alignment Rules (Context Rule upper onto alignment). State order (Concept event):
  + (Context : Rule, Kind, Concept : Rule, Kind);
  + Inference: Rules that emit Alignment Rules (ML / FCA).
  + Alignment Rules produced Statements:
    - (Employment, Employee, PromotionRule, Manager);
    - (PromotionRule : access Employment occurring Rule instance, EmployeeStatements, PromotionMapping, ManagerStatements);
    - (PromotionMapping : access PromotionRule occurring Rule instance, EmployeeResource, ReplacementRule, ManagerResource);
    - Apply Context Rule (onto mapping) in each step.
    - Returns updated Statements.
* Augmentation Rules. State order (Concept event):
  + (Context : Rule, Statement, Concept : Rule, Statement);
  + Statement Kinds: Alignment Rules.
  + Inference: Rules that emit Augmentation Rules (ML / FCA).
  + Augmentation Rules produced Statements:
    - Context : Relationship / Kind.
    - Infer Relations: Transitive, Reflexive, Symmetric.
    - CSPO Statements.
    - CSPO Populated Kinds.
    - (Context, CSPO, Concept : Rule, CSPO);
    - (Employment, peterEmployeeStatements, Promotion, peterPromotionStatements);
* Matching Rules. State order (Concept event):
  + (Context : Rule, Resource, Concept : Rule, Resource);
  + Inference: Rules that emit Matching Rules (ML / FCA).
  + Matching Rules produced Statements:
    - (Language, "Peter", sameAs, "Pedro");
* Spring Rest Repositories HAL Explorer
* FCA Implication confidence / support.
* Resource: getResourceOccurences. CSPOs / Kinds in context Statements.
* Rules: (semiotics context / concept) Grammar. Statements: Productions: Alignment Statements (ResourceOccurrences Rule Kinds). Terminals (ResourceOccurrence) / non Terminals (Kinds) / Primitives / Relationships (Context Concepts): state Matching Rules.
* (Context ContextKind, LHS ResourceOccurrence, Context Concept Kind / Context ResourceOccurrence Kind, RHS ResourceOccurrence);
* Apply Context function Kind to input ResourceOccurrences Flux : LHS. Apply Context Concept function Kind to LHS Flux : RHS.
* (Amor, (Pedro, amaA, Maria), amada, María);
* (Amor, (Pedro, amaA, Maria), Maria, amada);
* (Amor, Pedro, (Pedro, amaA, Maria), amante);
* (Amor, amante, Pedro, (Pedro, amaA, Maria));
* (Amor, Pedro, amante, (Pedro, amaA, Maria));
* (Amor, Maria, Pedro, amada);
* (Empleo, Maria, Pedro, compañera);
* (Familia, padre, hermano, tío);
* (Familia, hijo, (padre, hermano, tío), tío);
* (Empleo, (Pedro, trabajaPara, unEmpleador), (unEmpleador, brindaServiciosPara, unCliente), (Pedro, brindaServiciosPara, unCliente));
* (etc.: CSPO, Kinds, Statements). ResourceOccurrences LHS, Concepts (ResourceOccurrence Context Kind), RHS:
* (Statement, CSPO, Kind)
* (Statement, Kind, CSPO)
* (Statement, Statement, Statement)
* (Kind, Statement, CSPO)
* (Kind, CSPO, Statement)
* (Kind, Kind, Kind)
* (CSPO, Statement, Kind)
* (CSPO, Kind, Statement)
* (CSPO, CSPO, Kind)
* Rules Aggregation: Rule Context application matches / filters input Statements Flux for LHS Statements, Kinds, CSPOs Flux. Concepts Aggregated by Context CSPOs, Kinds, Statements. RHS result of applying Concept Kind to LHS Flux (infer Grammar).
* StatementKind : ContextKind;
* Ontology Matching: state (inferred) equivalence between types, instances, attributes, relationships and values. Rules / Grammar.
* Ontology Merge / Align: state (inferred) equivalence Statements and Upper Ontology mappings between types, instances, attributes, relationships and values. Rules/ Grammar.
* ContextKind: Aggregate Property Attributes (Employment), Object Values (Employeer). Context Resource: Subject Kind.
* Ontology Matching: Rules / Implications. Kinds instances, attributes, values. Order.
* Rules, Implications Statements: ordered Lists (subsumption / LHS domain, RHS range: extract SubjectKind from ObjectKind by inverse relationship). Comparator against Statements product: lt (sub / next), gt (super / prev), eq (aligned / matched).
* Rule Statement (Kinds) ResourceOccurrences: Implication Statements (Kinds CSPOs). Statements Resources: reified bnode, RDF\*.
* Apply Kind (Function) to Monad wrapped ResourceOccurrences. Statements: CSPO Kinds. CSPOs: Kinds occurrences. Kinds: Relationship (Concept).
* Contexts / ContextKinds: Implications / Rules (Upper asserted / Aligned Knowledge / Primitives). (Context, LHS, Concept, RHS);
* Implication / Assertion Statement: (Context, Subject, Property, ModelObject);
* ContextKind Aggregation. Instance: Context, Atribute: Concept, Value: RHS, from Assertion Statements.
* Rule Statement: (Context: ContextKind, LHS: SubjectKind, Concept: PropertyKind. RHS: ObjectKind.
* Aggregate Kinds into Rule Statements. (KindStatements). Statements match Contexts, match SK, apply Concept, match OK.
* Materialize Rule Statements from Model into CSPOs Occurrences from Resources:
* Rule: (Amor, Amante, Ama, Amada);
* Statements: (unAmor, pedro, amaA, maría);
* Rule: (Son, Father, BrotherOf, Uncle);
* Statements: (aSon, aSonFather, brotherOfFather, aSonUncle);
* Rules Alignment: RuleConcepts / RuleImplications. Aggregated from CSPO Occurrences, Kind Relationships, Statements / KindStatements Relationships. Rule / Implication: (Context, LHS, Concept, RHS); Populate Rules with Upper Alignment knowledge and aligned Model inferred knowledge.
* Rule / Implication: (Context, LHS, Concept, RHS) Inference: Kinds Instance, Attribute, Value CSPO Kinds Contexts aggregation / subsumption (domain / range flow). Matching: CSPO Kind Attributes matching those of Rules / Concepts.
* Relationship Statements: Rules (LHS, RHS in a Context by a Concept Property) stated via Upper Aligned types, instances, attributes, values Statements, Kinds, CSPOs and Resources (Concepts / Implications).
* Relationships / Rules Model. Model Aggregation / Inference.
* Implication (instances): Statements, Kinds, Resources. Parse instances as new Rules / Concepts.
* Model (Upper Aligned CSPOs):
* Rule / Implication: (Context, LHS, Concept, RHS);
* LHS, RHS : ResourceOccurrence (Statement / Kind / CSPO) wrapper Monad.
* Context, Concept : Kind. Rule / Implication. Rules reified by Kinds: Rule / RuleKind (Aggregated RuleKindStatements): (ContextKind, InstanceLHS, ConceptAttributeKind, ValueRHS). Stated / Inferred Rules / Implications RuleKindStatements.
* RuleKind: ContextKind; Instance: LHS ResourceOccurrence; Attribute: (Property) Kind, Value: LHS ResourceOccurrence. (wrapped ResourceOccurrences). RuleKinds / RuleKindStatements match.
* CSPO Contexts: Rules / Implications. ContextKinds: Contexts / Concepts. Initial Rules: input Statements. Aggregate RuleKinds (Aligned / Upper Populated RuleKindStatements).
* Kinds : Function::apply(Kind arg) : Kind (resolve instances, attributes, values occurrences).
* ResourceOccurrence wrapping Monad (Resource, Context, Kind), Kind Functions (domain / range, subsumption).
* Resolve Rule Context Kind Resources. Feed Rule LHS Resources.
* Apply Rule Concept Kind to previous Rule LHS Kind Resources output.
* Map Rule Concept Kind outputs into RHS ResourceOccurrence result.
* (Context, LHS, Concept, RHS); Upper / Aligned Primitives: (LHS, RHS). Primitive Contexts, Concepts.
* Rule Aggregation: Each Statement is itself a Rule stating a single fact building a Concept by means of its Kinds relationships / CSPO inter Statements occurrences.
* Initial Rules / Implications: initial Statements, KindStatements. Aggregate / Align Statements CSPO Kinds / KindStatements Rules / Implications by their attribute / value CSPO types Kinds.
* Rule: (Kind) Statement / Kind / Resource matching LHS (Upper onto matching aligned), (Upper / Model inferred) Concept in Context, (Kind) Statement, Kind, Resource RHS.
* State: Type according property values / property values relationships. Alignment, Augmentation: shapes (Upper OWL, SHACL, ShEx). Metakinds (person / adult: age value gt 21; brother / uncle: brother :hasChild). Order relationships (Statements). Activation (Context inter-resource relationships). Reification: express inter (Kind) Statements relationships. Statements ResourceOccurrence (Kinds / CSPOs) of Concepts Resources. Relationship Statements (upper / shapes).
* Upper Ontology / Matching: State implies Event / Relationship / Type: maritalStatus, uncle. Employee isA Worker. State facts by means of Upper types, instances, properties, values, relationships (RelationshipStatements: Statements about Statements, Resources, Kinds concepts).
* RelationshipStatements (Kinds / CSPO, Relationships (reified Statements). Aggregate inferred rules / logic statements (Resolvers / Notation3). Upper ontology types, instances, properties, values, relationships (sameAs).
* Upper Ontology Relationships: property values according type. before, during, after. Greater than, less than, equal. Event. Cause / Effect (state flow). Contains (Relationship, Place, Duration). Relationship kinds. N-ary Relationships (roles, actors templates).
* Augmentation / Alignment: Attribute / Links prediction. Upper Ontology alignment. Ontology Matching. Predicted Attributes Values.
* Activation: Kinds order (in Relationship Contexts).
* Order Statements by Kind hierarchies, previous / next common Attributes values.
* Label (add to Resource) Resolvers. RDFS#label NLP NER.
* Upper Ontology: ISO Topic Maps (TMDM, TMRM), ISO 15926 (OWL Templates).
* Statements / Kinds: order. Domain / range dataflows. Alignment, link prediction (between Kinds, KindStatements FCA).
* Activation: Kinds Functions (order: domain / range). Apply (map) adds (transition fits by schema) / query for Attributes / Values to Resource Kind Monad occurrence.
* Activation: Order Statements / Kinds transitions. Single, Married. Married, Divorced. Temporal order (octal comparison values encoding). Previous(Current / Next), Current(Previous / Next), Next(Current / Previous) Kinds.
* Inferred Attributes: Value Regression / Prediction (occurrences model). Scaling, MultiValuedContext.
* Inferred Attributes: Values prompts / placeholders (Resolvers).
* CSPO Kinds: Parameterized Instances, Attributes, Values.
* Kinds Impl: get inst (instances, instance), attrs(inst), vals(inst, attr) CSPOs.
* Kinds get Instance, Attribute, Value CSPOs. Fix / Move to CSPO Kinds.
* Input Statements: (Class, Instance, Attribute, Value). From RDF4J Model / Query.
* Query Repository rdfs:type value: Context, SPO.
* Kind Labels: from string similarity Subjects / Predicates NER.
* Reduce Context (FCA).
* Kinds hierarchies: Concepts subsumption.
* Activation: SubjectKinds / ObjectKinds: DCI Roles. PropertyKinds: Roles Behavior (Kinds Functions). KindStatements: DCI Contexts. Statements: DCI Interactions. CSPOs: Actor Occurrences.
* Alignment / Activation: Materialize Statements from KindStatements Kinds instances, attributes, values. FCA Objects / Attributes: Kinds.
* Activation: Rest (JAF), Restful Objects, HAL, Spring REST. HATEOAS.
* Alignment: Set KindStatement Kinds Kind.
* Aggregation: returns KindStatements. Kinds.getCSPOs.
* Scaled Attributes: Values.
* Kinds / Statements occurrences Resources, Kinds (roles), Statements (RDF star, reification).
* CSPO Statements.
* KindStatements
* RelationshipStatements (Kinds / CSPO, Relationships (reified Statements).
* KindStatement: schema Statements.
* Kinds Statements implements Kinds interfaces, Statement extends ResourceOccurrenceImpl (Context Statement Kinds: roles / metaclass: domain / range). CSPO Statements: Kind Statement Instances (Kind Statement getStatements).
* Reification Statements: example: SubjectKind implements Subject. Subjects SubjectKinds SubjectKindStatements.
* Kind Statements (Aggregated): SubjectKindStatement extends Statement. C: Super Kind, S: Kind, P: PK (domain), O: OK (range), From merged Occurrence Context Statement Kinds. Kind Statement Kinds: role / metaclass.
* CSPOs: Resource IRI / PrimeID. Merged Kinds: Instance, Attribute, Value Context Lattice products.
* Resource: Function and Monad wrapped value. Domain / range state / flow context / interactions order. Relationships. Activation.
* Statements URN: lookUp by CSPO hashCode (Map / Embedding).
* Statement IRI: Reification, RDF-Star (URN Encoded).
* Resource PrimeID, Resources URN Encoding.
* Kinds Aggregation: Merged Resource IRI (Instance, Attribute, Value URN). Prime factors ID. FCA4J.
* FCA Contexts:
* getContextAggregatedKinds
* getSubjectAggregatedKinds
* getPropertyAggregatedKinds
* getObjectAggregatedKinds
* FCA. Primes Embeddings. Property contexts, Subject objects, Objects attributes. Aggregate by Property contexts. Lattice Graph (Alignment).
* Serialize Aggregation, Alignment, Activation into RDF / RDFS / OWL / JSON-LD: Instantiate Model via RDF4J APIs. (Align upper ontologies / [schema.org](http://schema.org)).
* JSON-LD Serialization (ObjectMapper). Types documents: Kinds, Resources. AggregationService performAggregation.
* Convert any input into triples graph (Statements). Alignment: Graph Deep Learning). DOM Mappings. UIMA / Tika / any23 / D2RQ.
* Kinds: Initial Kind Resource: Kind Instance Resource. Aggregate: Instances Context (Attrs / Values).
* Target: Generate typed RDF RDFS / OWL (Kinds). Sets Aggregation.
* Target: Discover Links / Relationships. DOM Augmentation.
* Target: Discover Data Contexts / Interactions. DCI Activation.
* Kinds : Statement (CK Kind IRi)
* Kinds interface / impls: get / set Inst, Attr, Val (Statement ctx).
* getSubjectAggregatedKind()
* Resource / ResourceImpl occurrences / hashCode.
* CSPOs, Kinds, Statements: IRIs Occurrences roles. IRI in CSPOs, CSPOs in Kinds / Statements, Kinds in Statements. Resource get occurrences (Subjects, etc. streams / filter). Resource super type (IRI / Occurrences) CSPO / Kinds / Statement parameterized (Statement Resource<CSPOs, Kinds>).
* Statements: Model / Model Statements wrapper. Streams (load statements, aggregate kinds, populate CSPOs, filter / occurrences: CSPO / Kinds Statements). Statements (filter / occurrences set streams, Statement instances).
* Todos los atributos de un Subject son un Kind: CK (Kind Statements Stream). Kinds (filter / occurrences sets streams, Kind instances).
* Merge Kind Statements. Kind IRI: Statement Contexts.
* Populate CSPOs. Filter Statements Stream. Subjects, etc. (filter / occurrences set streams, Subject, etc. instances).
* Quads (Resource) : Type, Instance, Attribute, Value (parameterized). Interfaces.
* Resource : IRI, Occurrences (CSPOs, Kinds, Statements). Implementations.
* Context
* Subject (SK, IRI, Ps, Os)
* Property
* Object
* SubjectKind (Super SK, SK, Ps, Os) : P / O
* PropertyKind : S / O
* ObjectKind : P / S
* ContextKind : (S, P, O)
* Statement (CSPO, CK:SK:PK:OK Combinations, SK en CK: Ps / Os).
* Functions (Transforms) : Reactive.
* Getters: Type, Instances, Attributes, Values
* Getters CSPOs, Kinds, Statements Occurrences
* Alignment: super Kind infer sub Kind (superset) Attributes (Aggregation) / Values (Regression).
* Aggregation: Kinds (IRIs) / Instances (Statement Occurrences) Attributes / Values. Sets Model. Classification (no labels)
* Alignment: Types / Instances Relationships / Links. Kinds / Attributes. DOM Model (DAOs).
* Activation: Roles / State in Contexts / Interactions Relationships. DCI Model. Clustering.
* FCA Embeddings.
* Statements / Kind Statements : IRI (toString / hashCode). Reification.
* D2RQ Sample Dataset (Cinema).
* Aggregation: Materialize Kind Statements (RDFS / OWL).
* Debug Aggregation results.
* Retrieveable IRIs : JSON-LD. Alignment: [schema.org](http://schema.org).
* Alignment: sameAs.
* Context Kind: Context, Predicate, Object.
* Kind (role) in Occurrences / Context Kind Statements. DCI Context, DOM Alignment.
* SK: Subject, Predicate, Object.
* Debug: Aggregation. Sample relational dataset (D2RQ) FKs (inferred / Alignment). Browsable Aggregation output format: Statements, Materialized Kind Statements: RDFS / OWL (Semantic Web / SPARQL Browser).
* Reactive Functional / Async:
* Flat map each 3 aggregations from initial flows, each return next aggregation input. Flat map 3 aggregation groups.
* FKs. Objects / Subjects referred by IRI, Aggregated end types (Kinds / Alignment), Relationship Kind (PredicateKind). Star Schema (Dimensions, Facts).
* IRIStatementOccurrence / Kind getStatementOccurrences.
* Alignment: query / views / rules parameters: add OWL Templates to Repository Model. DOM DAOs.
* Materialize inferred Statements (reified Kind Statements) in Repository temporal Model. Alignment: SPARQL query templates / views / rules. RDF4JTemplate: DOM DAOs. Services.
* Materialize RDF rdf:class (Kinds), rdf:type, etc. Inferred attributes. Alignment queries.
* Materialize: Statements: Reification / RDF\*. Statement : rdfs:Statement. IRIStatementOccurrence. Statement / Kind Statement IRI.
* Handle Literals: Literals, Map<hashCode, Literal>. IRI: Literal type + hashCode.
* toString: Complete Entities / FKs JSON. Messaging: parse Resources toString. Autowired Custom ObjectMapper.
* Convert Aggregation to Functional / Reactive. Integrate with RDF4J.
* Aggregation. Learn Types.
* Alignment. Learn Relationships.
* Activation. Learn Interactions.
* Alignment: Functional / Reactive APIs. DOM / DAOs. Embeddings.
* Alignment Service: align(), get Types, Entities, Attributes, Values. RDF4JTemplate DAOs.
* Resource (HasIRI) interface. Resource::apply(Resource res) : Resources; Dynamic Functions.
* Map / flatMap Model domain (Static Functions).
* Zip T1, T2 (Resources product): apply T2 to T1 (Dynamic Functions) : Resources.
* Alignment: Events, Order relationship. Inferred.
* Update RDF4J with Alignment inferred Relationships (query / rules SPARQL).
* Emit inferred Statements.
* TODO:
* Kind has Attributes.
* Attributes has Kind according Occurrence (Instance).
* Attribute has Value according Instance.
* Values has Attribute according Occurrence (Kind).
* Replace placeholder IRIs with primes product URNs (Embeddings).
* Kind, Instance, Attribute, Value: IRIs / Embeddings.
* Primes.common(BigInteger, BigInteger).
* Primes.contains(BigInteger, BigInteger).
* Kinds Naming: Context IRI. KindStatements. Contexts Label.
* Aggregation: order. Kinds sub / super Kinds relationship encoded in Statement Context IRI.
* Kind reification: S : (ParentSK, SK, P, O);
* Alignment: Links Attributes / Values inference / materialization.
* Alignment: Theory. Order. State. Rules. Events. Relations. Uncle : (aFather, fatherBrother)
* Alignment: Attribute / Value clustering determines meta-Kind (age > 21 : Person age Adult Adult Person); (birthPlace BsAs, Person nationality Argentina, Argentinian Person).
* Alignment: Upper Ontology. Mappings.
* Alignment: Ontology Matching.
* Activation (DDD DOM / DCI). REST Dialog front end.
* Activation Service: activate(), get Interactions, Roles, Actors. RDF4JTemplate DAOs.
* Embeddings calculation (FCA / primes. Contexts / Attributes / Objects / Concepts.
* FCA Contexts: Axis, Objects, Attributes, Concepts.
* Restful DOM / OGM (DCI, CDI)
* Activation: Gestures (available Transforms).
* HATEOAS REST (HAL) Architecture. Components:
* Alignment / Activation DOM (RDF4JTemplate DAOs):
* Entity
* Name
* Type
* Value
* atributes : Set<Entity>
* Type : Entity
* Name
* players : Set<Entity>
* members : Set<Type>
* Role : Type (Wrapper)
* Type
* Declarative Behavior / Relations?
* Context : Role (Composite)
* Entity
* Type
* Role
* Declarative Behavior / Relations?
* Interaction
* bindings : Set<Context>
* Model:
* Statement : Resource
* Context : Context
* Subject : Subject
* Predicate : Predicate
* Object : Object
* Kind : Statement
* Kind<INST super Resource, ATTR super Resource, VAL super Resource>
* Context : Kind<> (this)
* Subject : INST
* Predicate : ATTR
* Object : VAL
* List<Kind<>> : statements
* Context : Resource (Wrapper)
* Statement : context
* ContextKind : role
* player : Resource
* ContextKind : Context, Kind
* extends Kind<Context, Subject, Predicate>
* Subject : Resource (Wrapper)
* Statement : context
* SubjectKind : role
* player : Resource
* SubjectKind : Subject, Kind
* extends Kind<Subject, Predicate, Object>
* Predicate : Resource (Wrapper)
* Statement : context
* PredicateKind : role
* player : Resource
* PredicateKind : Predicate, Kind
* extends Kind<Predicate, Subject, Object>
* Object : Resource (Wrapper)
* Statement : context
* ObjectKind : role
* player : Resource
* ObjectKind : Object, Kind
* extends Kind<Object, Subject, Predicate>