Meta Model:

Arrangement of layered typed CSPO quads statements in which each CSPO role / type plays the role of "aggregating" previous layer abstract knowledge into more concrete aggregated contexts (statements) instances until a "reference model" CSPO type arrangement is achieved, which is the root of all other layers hierarchy.

The purpose of this is to achieve some ontology matching capabilities over an upper abstraction set of layers and to enable a Functional Knowledge Base Interaction APIs integration / virtualization overlay for matching and consumption of distributed datasets via endpoints dataflows.

**Reference Model:**

ID : URL;

Occurrence: Context;

Context : ID (Context / ID : intension, Object / Occurrence, Sign / Kind / Metaclass / Attribute, Value / Role / Class : extension);

Root of Meta Model hierarchy.

**Notation:**

[LayerType] : [LayerSuperType] ([ContextType], [SubjectType], [PredicateType], [ObjectType]);

**Layers:**

Resource : Context (Resource, Resource, Resource, Resource);

Statement : Resource (Statement: c, Resource, Resource, Resource);

Role / Class : Statement (Role: b, Statement, Resource / Attribute, Resource / Value);

Role / Class aggregating CSPO Resource (IDs) sharing Attributes for their Objects / Values.

Kind / Metaclass : Role (Kind: a, Role, Statement, Resource);

Kind: Aggregated similar Roles occurring as Resources (Object) in Statements (Predicate).

Relation / Entity : Kind (Relation, Kind, Role, Statement: c);

Rel type (Relationship) instance / bindings. An Entity (Relation: intension) and their Statements for its Kind / Role occurrences (occurrences: kinds / roles Relation plays in statements. Matching. Object: extension). Data (DCI)[1].

Mapping : Relation (Mapping, Relation, Kind, Role: b);

Rel players types / bindings scenarios. Information. Interaction (DCI)[1]. Mapping Role and Relation Kind: dataflow promoted types / order: relationships players domain / range. Entity alignment.

Relationship : Mapping (Relationship, Mapping, Relation, Kind: a);

Rel type declaration, player types. Knowledge. Context (DCI)[1].

Mapping and Relationship layer contexts are "calculated" (reifying) by Relation layer context kinds.

Semiotic Layer (ontology matching):

Value (Value, Value, Value, Value);

Sign : Value (Sign, Value, Value, Value);

Object : Sign (Object, Sign, Value, Value);

Context : Object (Context, Object, Sign, Value);

Reference Model:

Root of MetaModel hierarchy.

**Matching / Relations / Attributes:**

One of the intentions of having all this layered infraestructure is to be able to inspect "relations", being them "reified" into a Relationship construct, or being them single attributes and values for a subject enabling the possibility of "align" one into another for ontology matching purposes.

(a, b, c: Kind, Role, Statement): Reified Rel. to / from expanded Attributes / Values. Matching / roles (intension / extension).

Context DOM: parent / child; previous / next siblings; attribute / value (determined by CSPO roles). Class / instance DOM relation for parent / children layers instances.

**Ontologies:**

Context layers instances. Levels. Example: Dimensional ontology. Ontologies should be able to be built upon Reference Model layer CSPO types arrangements.

Dimension, Unit, Measure, Value.

Axis, Behavior, Flow (state change), etc.

Primitives: dimensional upper ontology. In / Out, Prev / Next, Pick / Drop, etc. Opposites. State change (current). Events, state flows. Marriage example.

**Message Dataflow:**

Relationship, Mapping, Relation streams / signatures. Messages: Context instances. Functional Knowledge Base Interaction APIs.

Aggregation: Browse / Transform.

Alignment: Inference.

Activation: Dataflow type (signatures). Message dispatch (domain / range ordered). Aggregation.

**Relations:**

Inference. Relation types: transitive, reflexive, simetric. Campo, alcance, dominio, rango, transform / function: infer / aggregate. Context functor / monad.

Inputs: (Context / Relation, PK, column, value);

Inputs: (Infer S Kind / Role, S, P, O);

Inputs: aggregate occurrences. Statement Context for each SPO as Occurrence with corresponding Attribute / Value (S: PO, O: SP, P: SO, etc.).

Inputs (infer rels): Part / Whole. SPO / OPS. Attribute / Value.

Inputs (infer rels): Containment. SPO / SPO. Parent / Children. Occurrences of Contexts of same Context layers (inherited Contexts). Example: (Mapping, Mapping) for (Relationship, Mapping). Super / Sub type Contexts instances relationships.

Inputs (infer rels): Order. SO Ps Domain / Range.

Input (infer rels): Event. Prev / Next state change. Type promotion.

**Reactive Functional Reified Metamodel:**

Transforms: Match Selectors. Hierarchy polymorphism. Contexts streams. Browse Metamodel. Context, Subject Selectors.

Transforms: Templates. Context instances (CSs) declaration / augmentation (POs) Selectors. Metamodel activation. Predicate / Object Selectors.

Encode Match / Template as Context. CS: Match, PO: Template Selectors. Apply Templates (role bindings / prompts) declaration / augmentation activation to matching selected CS streams.

Augmentation: Aggregation, Alignment, Activation Reified Match / Template dataflows. Reactive Model instances Match / Template dataflows.

Selectors:

Apply Role to Statement : Statement / Statement to Role : Role.

Apply Kind to Role : Role / Role to Kind : Kind.

Apply Relation to Kind : Kind / Kind to Relation : Relation.

Apply Mapping to Relation : Relation / Relation to Mapping : Mapping.

Apply Relationship to Mapping : Mapping / Mapping to Relationship : Relationship.

Apply Context to Relationship : Relationship / Relationship to Context : Context.

**Meta Model / Backend:**

Reference Model / Occurrences annotations matrix:

Attachment: show an example of a fully expanded set of last model layer (Relation) and its corresponding occurrences property graph annotations. Previous model layers may be annotated accordingly in respect to their ability to aggregate more abstract contexts (Resource, Context) properties.

Reference Model / Occurrences annotations matrix:

Lattice / FCA: Contexts / Resources. Objects / Attributes (Contexts instances axes). (X, Y): Z (for corresponding pair types / functional transforms).

Statements and annotations: FCA Lattice / FCA Contexts / Attributes. Objects / Attributes (Contexts instances axes). (X, Y): Z (for corresponding pair types / functional transforms). Typed calculus in FCA context development. Sets.

IDs / bitstring encodings. Algebraic / arithmetic activation flows / templates / transforms metadata encoded selectors. Vector Space Model quads polygon embeddings.

Interfaces:

ID : URL;

Context: ID;

Object : Context;

Sign : Object;

Value : Sign;

Labeled Property Graph annotations example. Augments Reference Model. Statement example, Statement context aggregates SPO annotations (Statement occurrences data in other Reference Model layers contexts):

Context (Context : Object, Object : Sign, Sign : Value, Value);

Resource : Value (Resource, Resource, Resource, Resource);

Statement : Resource (Statement, Resource, Resource, Resource);

Role : Statement (Statement, Role, Resource, Resource);

Kind : Role (Statement, Role, Kind, Resource);

Relation : Kind (Statement, Role, Kind, Relation);

Relationship : Relation (Relationship, Role, Kind, Relation); \*

\*: Relationship: Aggregated Relation Statement Relation (Object) Roles / Kinds.

Hierarchy: render / process layers (Relation, i.e.) as Context, Resource, Statement, Role, etc. (upper layers) contexts (i.e.: reify Relation as Context, Resource, Statement, Role, Kind layer contexts). Context semiotic layer: aggregation, ontology matching.

Intension / Extension: S / O.

sub / super hiers, containment: P / O.

Relation reification: Relation statement object: relation instance. a: Role / b: Kind: relation ends. (a): Role reifying rel attrs / values. (b): Kind Resource reifying rel subject (rel players).

**Model:**

Message Events Bus.

Context Monad & type hierarchy (AST). CSPO parameterized types & aggregation (layers hierarchies specializations).

DOM:

Layer (Contexts): events producer / consumer (streams observer / observable).

DOM Parsing MetaModel: aggregate occurrences containment / hierarchies. Layers: subtype / supertype browse parent / children / siblings (order) and Attributes / Values relations.

Context::matchFilter(arg : Context): signatures / kinds stream predicate.

Context::applyMap(arg : Context): apply updates (CS Contexts / PO CRUD) matching filter predicate Context. Fires event bus messages.

Matching applies to meta-model signatures (internal aggregation, alignment and activation augmentations) and to domain / actual models signatures. Render new Attributes / Values and CSPO statements.

DDD: Declarative AST / Dataflow VM. Runtime. Encodings (layers / messages / activations). Event sourcing backends (Blockchain). Patterns (DCI).

RDFS / OWL / Graph Backend: Reified Metamodel. Labeled Property Graph. APIs.

Functional Meta Model Context Layers. Reactive streams. Match / Template Selectors. API.

Stream Resources Connectors (P2P Connector Bundles Context I/O). Dataflow. Backends / Augmentations / Endpoints model layers. APIs (reactive / events).

Layered abstraction levels streams options / menu semantics (REST). Forms / Flows. Browse, match / transforms high level APIs. HATEOAS CRUD / Flows.

Patterns. Input formats. Sample data.

**FCA / Concept Lattices**

Model context statements of an upper ontology and occurrence statements for each ontology concept occurrences using FCA (Formal Concept Analysis) and rules from a pseudo-grammar.

Context statements / Occurrence statements: Grammars. Concepts / objects hierarchies: CSPO statements concept types / kind rules / terminal instances. Productions: concept types / kind rules / terminal instances mappings / flows.

Aggregate kind rules (grammar) into context statements / Occurrence statements: polymorphically, Kind context applies to all Resource hierarchy (all lattices).

Contexts parsing: monadic parser combinators / monadic AST. Recognize context types from (surrounding) reified kind types / rules (link grammar). TBD.

Rules are of the form:

(TypeA, TypeB) > AggregatedKindResources;

For example, in Relation lattice:

(someRoleA, someKindB) > AggregatedRelationResources;

Dispatch: Model (lattices) observer / observable (streams / functional) of grammar (contexts case match) events.

Lattice (FCA Contexts) population and Augmentation:

Base layer: Context. Resource, Statement, Role, Kind, Relation layers.

Input layer: case match layer grammar type signature. Reactive dataflow dispatch inputs top-down / bottom-up contexts layers hierarchy (lower hierarchy layers polymorphically materializes upper hierarchy layers) for Augmentation.

Lower hierarchy layers contexts mapping / function transforms into next upper layer context (example: Roles to Kinds). TBD.

Contexts (TBD):

Data layer: Resource, Statement, Role, Kind, Relation contexts (aggregation).

Information layer (occurrences / interactions): Data layer contexts statements products as new contexts (aggregation).

Knowledge layer (Dimensional / DCI Contexts): Information layer contexts statements products as new contexts (aggregation).

Tensor like arrangements by FCA / grammars (TBD):

Aggregations: Data layer contexts containing Information layer contexts containing Knowledge layer contexts. Encoding: embeddings, object / attribute features bitstrings, VSM (Vector Space Model) quad polygon angles. Order encoding (lattice). Activation / Aggregation (CUD): preserve order (lattice), arrange input message as production of corresponding context shape. Retrieval / transforms: resolve production of corresponding context shape for input message.

Augmentation:

Activation: Layer receives matching context message.

Aggregation: Layer resolves productions of grammar rules for context message.

Alignment: Resolution of relevant knowledge and input message. Emits output message.

Notes:

Context occurrences statements (i.e.: Statement in Kind context). Occurrence contexts: (S, S); (P, P); (O, O);

FCA: Lattice. Ordered Sets. Intension / Extension.

To Do:

Model Order. Axes: dimensions, units, measures. Events (measures / relations / state boundaries in data / information / knowledge levels: price, price at moment in time, variation, tendence predictions, idem for distances, etc.).

Hierarchies: metaclass / role, class, instance, occurrence (parent, children, previous, next, attribute, value). Encoding. Comparisons. Functional traversal (streams).

Dimensional contexts: Contexts from Occurrences contexts statements. Dimensional contexts: Events (attributes). Order relations assertions by context occurrences hierarchy domain / range, set / superset attributes relations.

(Mapping, Kind, Role, Statement);

Event (Dimensional context attributes): (Mapping / unit / class, Mapping super / parent / dimension / metaclass, Kind unit / measure / occurrence, Role measure / value instance);

Model Application domains: upper ontology (Behavior, Flow, etc.) encoded in meta model and specialization levels for domains contexts. Declarative abstractions ontology for application design: discovery, alignment and matching for services renderings and integrations.

Hierarchy: render / process layers (Relation, i.e.) as Resource, Statement, Role, etc. (upper layers) contexts (i.e.: reify Relation as Resource, Statement, Role, Kind layer contexts).

Context (reified / instances) kinds (topics): Resource content types. Resource (monad): representation, HATEOAS, dialogs. Dataflows (order / domain / range kinds).

HATEOAS: Form / Flow. Operations / Dataflow Representation / State IO (CRUD) prototypes / templates. Prompts (values / operations). Dialog. Gestures. Context: navigation state (i.e.: pick operation value prompt shows value type Form). DDD DOM.

**Meta Model (DCI)**

DCI Meta Models:

Data (data): Relation hierarchy Model.

Context (schema / dataflow): Augmentation hierarchy Model. Model layers extends corresponding Data layers.

Interaction (behavior / services) Dialog hierarchy Model. Model layers extends corresponding Data layers.

Contexts and Interactions Models extending / reified as Data Model layers enabling matching, inference and augmentations (FCA / ML embeddings for example) for behavior and schema alignments.

**Meta Model (Data)**

**Reference Model:**

ID : URL;

Occurrence: Context;

Context : ID (Context / ID : intension, Object / Occurrence, Sign / Kind / Metaclass / Attribute, Value / Role / Class : extension);

Root of Meta Model hierarchy.

**Notation:**

[LayerType] : [LayerSuperType] ([ContextType], [SubjectType], [PredicateType], [ObjectType]);

**Layers:**

Resource : Context (Resource, Resource, Resource, Resource);

Statement : Resource (Statement: c, Resource, Resource, Resource);

Role / Class : Statement (Role: b, Statement, Resource / Attribute, Resource / Value);

Role / Class aggregating CSPO Resource (IDs) sharing Attributes for their Objects / Values.

Kind / Metaclass : Role (Kind: a, Role, Statement, Resource);

Kind: Aggregated similar Roles occurring as Resources (Object) in Statements (Predicate).

Relation / Entity : Kind (Relation, Kind, Role, Statement: c);

Rel type (Relationship) instance / bindings. An Entity (Relation: intension) and their Statements for its Kind / Role occurrences (occurrences: kinds / roles Relation plays in statements. Matching. Object: extension). Data (DCI)[1].

Mapping : Relation (Mapping, Relation, Kind, Role: b);

Rel players types / bindings scenarios. Information. Interaction (DCI)[1]. Mapping Role and Relation Kind: dataflow promoted types / order: relationships players domain / range. Entity alignment.

Relationship : Mapping (Relationship, Mapping, Relation, Kind: a);

Rel type declaration, player types. Knowledge. Context (DCI)[1].

Mapping and Relationship layer contexts are "calculated" (reifying) by Relation layer context kinds.

Semiotic Layer (ontology matching):

Value (Value, Value, Value, Value);

Sign : Value (Sign, Value, Value, Value);

Object : Sign (Object, Sign, Value, Value);

Context : Object (Context, Object, Sign, Value);

Reference Model:

Root of MetaModel hierarchy.

**Dataflow Model (Context).**

Dataflow:

Iterations (types / kinds order declaration). Streams: Contexts / Occurrences Bus (signatures / discovery).

Conditionals (predicates / filters on types / kinds attributes / values) on Iterations.

Jumps (aggregation / stack sub-streams) on Conditionals. Apply Kinds on matching / referring Contexts (Employment, Person: Employee). Extract Kinds on matching / referring Contexts (Family, Father).

Order. Comparators: common upper hierarchies, Dataflow domain / range, SortedSet. Lattices (FCA contexts).

Augmentation: Aggregations, Alignments, Activations. Perform encoding dataflow.

Encoding: Augmentation, Template, Mapping, Transform.

OntResource: Ontology Matching (aligned URLs): semiotic context.

Dataflow Model:

Resource: (OntResource, OntResource, OntResource, OntResource);

Statement: (Transform, OntResource, OntResource, OntResource);

Role: (Mapping, Transform, OntResource, OntResource);

Kind: (Template, Mapping, Transform, OntResource);

Relation: (Augmentation, Template, Mapping, Transform);

Template: Reified Model and Model instances (hierarchies). Match inputs. Iterations.

Mapping: Dataflow reified operation flows bindings (subscriptions). Predicate / Object stream endpoints?. Conditionals.

Transform: Dataflow reified results. Jumps.

**IO / Services Meta Model (Interaction):**

Model for back ends synchronization and services exposures.

Context: (Model, OntResource, Resource, Resource); Key / value dictionary for source Model ontology matching.

Resource: (OntResource, OntResource, OntResource, OntResource); Full matched Resource descriptions: Type, ID, Attributes, Values.

Statement: (Assertion, OntResource, OntResource, OntResource); Transform (Jumps). Source Model(s) interaction interface Services URLs (IO).

Role: (Prompt, Assertion, OntResource, OntResource); Mapping (Conditionals). Predicates (LHS: Predicate, RHS: Object).

Kind: (Interaction, Prompt, Assertion, OntResource); Template (Iterations). Aggregate matching Interaction Assertion Prompts.

Relation: (Dialog, Interaction, Prompt, Assertion); Augmentation. State flows.

**FCA / Lattices**

For each layer context statement build tables which axes correspond to each context CSPO context types. Aggregate CSPO types / values in the form shown below.

FCA Context (tables): Context (Relation, Kind, Role, Statement, Context, etc.) matrices w./ corresponding CSPO x CSPO types axes. Types intersections determine cell type (as in example tables shown). Values intersections are instances of corresponding types. Example:

(RoleA x StatementB: KindC) : Kind RoleA plays in StatementB.

Thus, each layer context statements are used to build a matrix of CSPO x CSPO of its types and (scaled) values. In this manner (aggregating matrices / FCA contexts), SPO Resource occurrences in matrices axes SPOs / context layers statements SPOs are reified in statements for which the occurring SPO Resource is the statement Context and its SPOs are resolved according its SPO position in the original Context. This statements form the matrix rows and columns SPOs according some aggregation layout.

The purpose of this is to retrieve enough concepts (FCA) metadata to populate concepts / objects / attributes conforming a Lattice of related Resources and those relations values (as in the above example).

FCA Lattice (concepts / attributes / objects): (types / values) x (types / values). Encoding (IDs): ontology matching enabling type / instance calculations / traversal / transforms.

Relation matrix:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Relation | Kind | Role | Statement |
| Relation | Relation | Kind | Role | Statement |
| Kind | Kind | Relation | Statement | Role |
| Role | Role | Statement | Relation | Kind |
| Statement | Statement | Role | Kind | Relation |

Matrices for other layers (Kind, Role, Statement, Resource, Context) contexts follows the same principles.

Reifying one aggregated layer SPO layer (for example: Kind in the previous table) has original context matrix axes in the corresponding SPO layer (Subject in this case):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Kind | Relation | Statement | Role |
| Kind | Kind | Relation | Statement | Role |
| Relation | Relation | Kind | Role | Statement |
| Statement | Statement | Role | Kind | Relation |
| Role | Role | Statement | Relation | Kind |

The “generic” form of the table is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Context | Occurrence | Attribute | Value |
| Context | Context | Occurrence | Attribute | Value |
| Occurrence | Occurrence | Context | Value | Attribute |
| Attribute | Attribute | Value | Context | Occurrence |
| Value | Value | Attribute | Occurrence | Context |

Use cases:

Use FCA Lattice for sorting / ontology matching / augmentations / query / ontology browsing.

Aggregation: Complete contexts objects / concepts / attributes by FCA / inference.

Inference example: (Statement x Statement): Relations between both Statements.

Learning: ML embeddings for types / values / concepts.

TBD: (metaclass, class, occurrence, instance) relations / atttributes.

TBD: Set oriented intension (C) / extension (O) and relations between sets.

TBD: Discover IDs / encoding techniques enabling algorithmic translation of models operations.

Encoding: FCA Scaling. FCA Context objects and attributes are corresponding CSPO Contexts types scaling enclosed Context types instances. A potential encoding of axes objects and attributes (rows and columns) would be a bitstring of length 4 x n, being n the length of an instance identifier for each quad Context encoded in its corresponding bitstring quad space (4 is for CSPO quad types instances identifiers segments). Then, navigation should be allowed from a pair of object / attribute to another object / attribute: (type, object) x (type, object): (type, object).

If Context types / instances identifiers are sequential in form, a mapping (hashing) could be done in a bitstring of the length of scaled attributes (columns) having a 1 in the corresponding attribute for a given object. Attributes also may be rendered as a sequence of prime numbers being an object extension the product of its attribute primes.