Application:

Workflows (Domain Goals) general purpose ontology matching integration framework.

* Products And Services Community Exchange Network. Resource Oriented Knowledge Computing. Purpose driven Needs / Goods / Products Goals interactions.
* Data, Information, Knowledge exchange: data / schema / behavior Augmentation of virtualized and syndicated / aligned business domains. Business domains applications purpose / problem "spaces" interactions / translations. Addressable interactions: event sourcing.
* Purpose modelling: Business Domains.
* Semantic Hypermedia Browser: declarative front-end / services. Forms / Flows. Annotation, Augmentation & other Domains.
* Workflow Domain. Layer Contexts Domains.
* Workflows (Domain Goals) general purpose ontology matching integration framework.
* Levels: Upper / Onto Matching: reify Resource upper layers as Resource and aggregate into lower layers. Reified Entity, Relationship, Flow, Domain as upper layers and aggregated downwards (Rules / Productions). Productions dataflow (domain / range). From UI Gesture to backend operations.

Reference Model:

Encodings:

(Type, Object) Key / Value.

(Context, Subject);

(Subject, Predicate);

(Predicate, Object);

Reference Model: Key / Value de-referenceable (for matching / embedding purposes) URIs having as host the peer that identified the Resource. DIDs resolution. Cons cells encoding for Domain Component Model I/O.

* Quads Encoding: (Context, Occurrence, Attribute, Value). Kinds. OGM. Sets / FCA. Context, Metaclass / Role, Class, Instance, Occurrence (Context).
* Matching: Addressable / Browseable Encodings (FCA / Sets) / Identifiers. Order, Dimensional / hierarchical relations (attribute sets). Root Layers (Reference Model) traversal.
* Functional: Selectors.

Domain Component Model:

Configures runtime (object / functional Domains) for Message parsing and behavior executions.

Domain Declarations: populate layers from Semiotic Context layer Domain description layers resources: Service Resources I/O layers matching / producing semiotic statements for Domain I/O.

Model: Triple Store. Meta Model Schema (RDF / RDFS). Upper Ontology. Primitives.

Controller: Resource Layers object hierarchy API. Named Transforms (Resource URI Service interface / implementation bindings) dataflow: signatures pipelines. Triple Store object graph (DTOs).

View: Layers Domain hierarchy functional API. Messages I/O. Protocol.

Bus. Messages / Layers I/O.

Triple Store:

Meta Model Hierarchy / Schema (Quad Layers / DOM):

CellValue : Value

ColumnField : Sign

ID : occurrence (PK) : Object

Context : instance (table) : Context

Role : metaclass (CSPO)

Resource : class. Monad Value (instance)

Kind : selector / transform (Functor mapping). Monad Value Type (metaclass / role)

Statement (context)

Relation : Kind Grammar (Productions). Monad Instance (occurrence)

Entity : Kind Grammar (Rules). Monad Type (class)

Relationship

Flow

Domain

Resource Layers object hierarchy API:

Functional Layers Domain model / transforms (events / controller). Named Transforms (Resource URI Service interface / implementation bindings) dataflow: signatures pipelines. Triple Store object graph (DTOs).

Value

Sign

Object

Context

Role

Resource

Kind

Statement

Relation

Entity

Relationship

Flow

Domain

* Meta Model: Layers.
* (Context / Relation, Object / Statement, Sign / Kind, Value / Resource).
* Augmentations:
* Aggregation: Rules (Objects).
* Activation: Rules (Signs).
* Alignment: Rules (Values).
* Layers. Roles. Functions (mappings).
* Upper / Lower Layer Roles. Transforms. Bindings (contexts resolution by reference model matchings).
* Intension / Extension. OntResource: Resource reifying aligned data / schema / behavior (data / information / knowledge.
* Type Promotion (roles). Order (dataflow). Reified Relation / Relationship (Production / Rule) context roles / interactions. Matching.
* Layers Resource Context / DOM API. Levels (inheritance hierarchy reification).

TBD.

Layers Domain hierarchy (Functional API):

Monads AST / Parser Builder. Monads: Parsing / Matching, Zippers. Introduction. API: Augmentations, Transforms / Mappings. Traversal. Dataflow.

Layers Monads / Parser Monads (Messages : Rules / Productions). Functional events dataflow (selector signatures : Layer instance Activation).

DOM / AST Hierarchy:

Resource<OntResource[]>;

Kind<Resource[]>;

Statement<Kind[]>;

Relation<Statement[]>;

Entity<Relation[]>;

Relationship<Entity[]>;

Flow<Relationship<Entity[]>;

Domain<Flow[]>;

* Functional Model Monads wraps Meta Model Layers DTOs which represents an endpoint / interface for its Resource URI type and instance.

Bus. Messages / Layers I/O:

Reference Model encoded Messages. Domains streams / pipelines. Messages enter Domains layer through Triple Store and returns back (results, prompts / populated). Messaging backend.

Messages:

Reference Model: Matching (recursive) of aggregated Rules (keys) Productions (values). Domains matches from upper to any level of (inter) Domain activations.

Layers: Productions in one layer are Rules of next layer (context, class, metaclass, instance, occurrence, role, etc.).

Messages flow from Domains layers matching Rules and producing further Productions (Messages) matching lower layers Contexts (Rules) until Service Resource (semiotic) Domain layer (Transform: URI service implementing interface). Transform semantics goes back until Domain layer: Augmentation of Resource layers (prompts / protocol semantics).

Message Layer : Semiotic (Domains) Layer.

Message matching on Layers following aggregated Contexts (resources / selectors / prompts) until Domain Semiotic interaction. Response Messages augmented into aggregated Domains response.

* Message: Meta Model Resource Hierarchy (Reference / Semiotic Model) root Layer. Resource Set specification (Rules / Productions). Further Layers implement Message (matching) behavior (polymorphism). Value in role / type / occurrence / context Resource (prompt). Data / schema / behavior I/O.
* Message: Perform Augmentations as with source Domains data (Message Layer Context type, Prompts / Assertions: TBD).
* Aggregation: Match Layers Contexts (Rules Productions).
* Activation: Match Resource Kinds.
* Alignment: Match Resource Values.
* Dataflow: streams, domain / range, endpoints, routes, pipelines. Message I/O (prompts: model / client).
* Bus: Message dispatch. From Aggregation Dataflow matching context subject, kind, resources. Semiotic layer encodes domains graph dataflow.
* Meta Model parsing (by Functional API): Layer Rules (inference prescriptions) / Layers Productions (inference propositions). Parse Message as corresponding Layer Context (for which Message proposition is true for prescriptions).
* Messages:
* Rules: Entity (grammar: infer available messages).
* Productions: Relation (infer / parse Messages).
* Matching: Kind (selector : Entity Relations).
* Transforms: Kind (transform: Relation Productions of Entity Kinds).
* Dataflow: Result Transform matching rules signatures.
* Layer::flatMap(attr : Kind) : Layer;
* Message: Graph layer statements(s) populated with Relation (Productions) nested into Entity Rules to be applied / applicable to the Message Relations (Productions). Relations with concrete Resources or Kind matching model layers instances. Existing or new Production: Resource or empty Kind results. Update / Delete: override previous version.
* Build Message graph via navigation of the model (Forms / Flows HATEOAS APIs, Kinds domain / range). Transform mapping: Kind prompts: apply Rule Kind to Production Resource: Productions.
* Message: Relation statements. Productions.
* Message: Rules. Relation statements of Rule Kinds on to operate over Relations of Kind.

Augmentations:

* Augmentations Domain:
* Aggregation: clustering stream. JNDI. Registry. Roles in Contexts.
* Activation: classification stream. JAF. Naming. Types in Contexts.
* Alignment: regression stream. Index. Attribute Values in Contexts.
* Model reification: Role Context. Addressable Augmentations (Object extension which is result of Context intension).
* Kinds: Streams of corresponding Roles.
* Semiotic Layer: (Augmentation, Subject, Predicate, Object);
* Aggregation Rules (Layers / Kinds / Resources Mappings):
* (Context, Occurrence, Attribute, Value);
* Map Reduce (Reference Model). Object occurrence of Predicate.
* Reference Model: Map Reduce. Reified Layers. Levels.

Component Domains:

Functional event driven Domains configures models behaviors. Domain types: service resources, interfaces (transforms), signatures (dataflow).

I/O / Persistence Domain.

Sets Augmentations Domain.

FCA Augmentations Domain.

Endpoints I/O Domain.

Predictions Domain.

Dimensional Domain.

Registry Domain.

Index Domain.

Naming Domain.

Business Domains: business specific domain types.

I/O / Persistence Domain:

Events (event sourcing). Backends. Peers. DIDs.

Semiotic (Functional Message Signature): (PersistenceType, PersistenceSubject, PersistenceMember, PersistenceValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain persistence resource types (employee).

Member Kind: persistence members resource types (employee/salary;ARS).

Value Kind: range resource types (salary;ARS).

Reify Persistence as Relationship (Values as Relation Resources). Align domain / range with domains / primitive types (Member Kind, salary;ARS).

Event sourcing:

(PersistenceContext, PersistenceContext, PersistenceMember::new, PersistenceSubject);

(PersistenceContext, PersistenceContext, PersistenceMember::delete, PersistenceSubject);

(PersistenceContext, PersistenceSubject, PersistenceMember::delete, PersistenceSubject);

Sets Augmentation Domain:

Semiotic mappings population. Augmentations: Aggregation (layers), Alignment (ontology), Activation (layers dataflows). Render Resource hierarchies.

Semiotic (Functional Message Signature): (SetContext, SetParent, SetSubject, SetChildren);

Context Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Parent Kind: domain resource types (resource).

Subject Kind: attribute resource types (resource/resource).

Children Kind: range resource types (resource).

Reify Sets as Relationship (Values as Relation Resources).

Sets API: Augmentations, Transforms / Mappings. Traversal (Context, Statement, Kind, Resource).

FCA Augmentations Domain:

Semiotic mappings population. Augmentations: Aggregation (layers), Alignment (ontology), Activation (layers dataflows). Objects / attributes objects / concepts traversal. Render Resource hierarchies.

Semiotic (Functional Message Signature): (FCAContext, FCASubject, FCAAttribute, FCAValue);

Context Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain resource types (resource).

Attribute Kind: attribute resource types (resource).

Value Kind: range resource types (resource).

Reify FCA Context as Relationship (Values as Relation Resources).

* FCA API: Augmentations, Transforms / Mappings. Traversal (Concepts, Objects, Attributes, Products).
* FCA Contexts:
* Reference Model.
* Meta Model.
* Encodings.
* Domain Models Augmentations.
* FCA / VSM (Vector Space Model) Meta Model Context Encoding:
* Attributes: Resource URIs. Polygon side lengths (class).
* CSPO Roles (scaling): polygon sides (metaclass).
* CSPO scaling: ordered side position.
* Polygon sides dot-notation ordered sides lengths: Resource Layer Statement IDs (instance).
* Sides dot-notation sum: side in context (occurrence).
* Normalization: Resource URI attributes embeddings / primes quad polygon sides lenghts.
* Nested Resource encoded attribute values (layers hierarchy): sides lengths concatenation (ordered dot notation) sum (occurrence).
* Graph navigation (layers / transforms: concepts / objects containing / contained in concepts / objects attributes IDs / lengths).
* FCA Contexts. Encoding. Flows. Order (types: dataflow signatures domain / range, instances: dimensional attributes).

Endpoints Domain:

Streaming I/O Dataflow.

Semiotic (Functional Message Signature): (EndpointContext, EndpointSubject, EndpointRequest, EndpointResponse);

Context Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain resource types / referrer (employment).

Request Kind: request resource types (person).

Response Kind: response range resource types (employee).

Reify Endpoint as Relationship (Values as Relation Resources).

Augmented Resources Contexts / Interactions Services.

Forms / Flows (Grammar / Protocol Builder. Prompts). Resource augmentation endpoints. Forms / Flows browsing APIs. DCI: Declarative Forms / Flows.

OGM / Client Drivers Services.

* REST: Current / referrer. Rel. HREF. Link body. Metadata. Endpoint Domain.
* Monads: Reify available Transforms as activable Resources (Function addresses). REST / HATEOAS HAL.
* Prompts / Dialogs: Function arguments (values / options) shown as link addresses in Transforms navigation Flows. Activation browse of Resources in Transform context.
* Chained Activations for complete contexts resolution / flows. Complete layers productions rendering / navigation from higher to lower layers.
* Browse: request address content representation (extracted from current state) embedding current state representation as request context body. Model matches address and returns augmentation using request body as argument / context.
* Transforms (Resource functions transforms mappings) populated with possible source / dest values from context concepts objects / attributes. Concepts flows available when Resource matches source attributes. Transform Resources available for each DOM layer.
* Transform application merges / translates Entity Relations with applicable mappings from Resource functions transforms mappings (of which concept Resource has source / destination transform mappings).
* Protocol. HATEOAS. Available transforms flows rendered as browseable Resources. CRUD: Browsing values for a Resource mapping transform has REST semantics for activating concepts with new Resources.
* Selectors: matching / activation: match Monads functors by their attributes (signatures), apply transform (mapping function resource transform request address) over referrer body (yields next state functor). Events: Monad functors listen matching browsing events and publishes transform results.

Predictions Domain:

Semiotic (Functional Message Signature): (PredictionType, PredictionSubject, PredictionItem, PredictionValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain resource types (image).

Item Kind: prediction resource types (image/face).

Value Kind: range resource types (face).

Reify Prediction as Relationship (Values as Relation Resources).

Dimensional Domain:

Semiotic Layer: (DimensionType, DimensionSubject, DimensionItem, DimensionValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O. Time example (contains / before).

Subject Kind: domain resource types (hour; dayOfWeek) : 1. Monday.

Item Kind: dimensional resource types (hour/minutes; dayOfWeek/dayOfWeek) relations: contains / before.

Value Kind: range resource types (minutes; dayOfWeek) : 60. Tuesday.

Data / Key Value: Price.

Information / Schema: Tuples. Price variation.

Knowledge / Behavior: Values relations. Monthly price increase.

Reify Dimension as Relationships (Values as Relation Resources).

Upper Ontology: relations / primitives.

* Dimensional: inference / relation types / restrictions.
* (Context, Occurrence, Attribute, Value);
* (Context, Class, Metaclass, Occurrence); Instance, Role.
* Encode order / hierarchies / relations (parent / child, prev / next, etc.) / iterations / conditionals / jumps.
* Dimensional Domain: units, measures, values. Comparisons, relations. State. Events (marriage example). Verbs (action, passion, state). Order (data / schema / behavior).

Registry Domain:

Key / Value for graph contexts, nodes, predicates.

Semiotic (Functional Message Signature): (RegistryType, RegistrySubject, RegistryKey, RegistryValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain node resource types (person).

Key Kind: registry resource types (person/age;int).

Value Kind: range value resource types (age).

Reify Registry as Relationships (Values as Relation Resources). Align domain / range with primitive types (Key Kind, age;int).

Index Domain:

Indexing of graph contexts, nodes, predicates.

Semiotic (Functional Message Signature): (IndexType, IndexTerm, IndexScope, IndexValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Term Kind: domain node resource types (resource).

Scope Kind: dimensional resource types (resource/resource).

Value Kind: range value resource types (resource).

Reify Index as Relationships (Values as Relation Resources).

Naming Domain:

Terms translation in contexts for graph contexts, nodes, predicates. Alignment / matching.

Semiotic (Functional Message Signature): (NamingType, NamingSubject, NamingContext, NamingValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain node resource types (term).

Context Kind: dimensional resource types (term/term).

Value Kind: range value resource types (term).

Reify Naming as Relationships (Values as Relation Resources).

Domains Dataflow:

Layers Dataflow: Augmentation. Rules / Productions matching (Reference Model / Kinds Aggregation).

Semiotic Dataflow: Object Kind matches Subject Kind of Context Kind signatures. (Sucessive Layers Dataflow).

Ontology Matching:

Upper Ontology. Grammars. Primitives.

Matching: Resource occurs as context / occurrence / atribute / value or class / occurrence / context / metaclass / instance in equivalent occurrence contexts (kinds / order / shapes / type hierarchies).

Meta Model encodes mappings for equivalence / relations hierarchies for entities instance occurrences in roles in contexts for concepts recursively till upper onto / primitive terms / relations.

Reify relation from / to predicates (semiotic) / relation entity (expanded relation entity / roles statements). Dimensional measures / state events. Shapes: transforms / rules.

FCA Ontology Matching: Upper ontology / primitives. Reference Model objects / attributes encoding. Encoding (scaling): lattice concepts relations / transforms traversal.

Semiotic mappings population. Augmentations: Aggregation (layers), Alignment (ontology), Activation (layers dataflow transforms: context products).

Reference Model Contexts.

Meta Model Layers Contexts.

* Alignments (Reference Model types / values):
* Data Alignment: key / val.
* Schema / Information Alignment: tuples.
* Behavior / Knowledge Alignment: dimensional.

Appendix:

FCA Domain Contexts:

Semiotic (Functional Message Signature): (FCAContext, FCASubject, FCAAttribute, FCAValue);

Reference Model Context:

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Value | Value | Value | Value |
| Object | X |  |  |  |
| Object |  | X |  |  |
| Object |  |  | X |  |
| Object |  |  |  | X |

Meta Model Context:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Resource A | | | |
|  | Context | Subject | Predicate | Object |
| Resource B | X |  |  |  |
| Resource C |  | X |  |  |
| Resource D |  |  | X |  |
| Resource E |  |  |  | X |

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).

Layers Context:

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 (nested contexts).

Values intersections are instances of corresponding types. Example:

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

Rules are of the form:

(TypeA, TypeB) > AggregatedKindResources;

For example, in Relation lattice:

(someRoleA, someKindB) > AggregatedRelationResources;

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.

Nested Contexts:

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 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).

Layout: The aggregated statements have as Contexts the occurring SPOs in a Context layer statement and its SPOs are the occurrence Context and the other SPOs in the occurring statement. For a Context in an aggregated statement occurring as (SPO) in the occurrence statement, occurrence statement Context is its (SPO) and its aggregated (SPO) is occurrence statement (SPO).

Layout: Having a Context layer, a matrix (FCA context) of the form (CSPO x CSPO) is built for aggregation of models. The aggregated statements (rows / columns) have as Contexts the CSPOs (occurrences) of an axis and and its SPOs are given from the types / values of the context layout. For an aggregated Context statements / matrix, the original context is located in the (SPO) axis from the (SPO) which it was taken from the original Context matrix.

Augmentations: TBD.

Aggregation:

Activation:

Alignment:

Transforms (products / encoding) Dataflow:

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.

Sets Domains:

Semiotic (Functional Message Signature): (SetContext, SetParent, SetSubject, SetChildren);

Augmentations:

Aggregation:

Kinds: (Context : Kind, Resource, Attribute, Value);

Hierarchies: Resource reified Kind as Kind Resource (sub Kind).

Kinds layouts:

(S, P): OK; (P, O): SK, (O, S): PK; (SK, OK): CK;

Role: Sets. Layers CSPO Resource types.

Layers Aggregation: TBD.

Activation:

Resource Context: Kind.

Alignment: Kind Attribute / Value Resource Statements. Shapes (inference of Attribute Value by context, class, metaclass, instance occurrences.

Transforms. Dataflow.

Sets Layout and encoding bitstring mask format:

