**MODELS:**

**Reference Model**

Underlying Model for main persistence in the RDF store, reifying other models knowledge and enabling conversion back and forth other models representations handled by the Model Service.

Classes:

ID​

- primeID : long​

- urn : string​

- occurrences : IDOccurrence[]​

- CPPEembedding : long

IDOccurrence extends ID​

- occurringId : ID​

- context : IDOccurrence​

Statements:​

Data: (IDOccurrence(ID), IDOccurrence(ID), IDOccurrence(ID), IDOccurrence(ID))​

Schema: (ID, ID, ID, ID)

Implement Reference Model as an underlying core model for other models alignment / synchronization / persistence (Reference Model views) and services message exchange (schema / data statements) alignment format.

Property Graph: IDOccurrence instance A as occurringId of IDOccurrence instance B, B is occurrence context of A.

**FCA Model (Reference Model View)**

Classes:

Context

Relation

Object

Attribute

Statements:

(Context, Relation, Object, Attribute).

CPPE / RCV inference schema / data Statements.

**Sets Model (Reference Model view)**

Classes:

Context extends IDOccurrence

Subject extends IDOccurrence

Predicate extends IDOccurrence

Object extends IDOccurrence

Interface Kind<OccurrenceType, AttributeType, ValueType>​

- superKind : Kind​

- attributeValues : Tuple<AttributeType, ValueType>[]

- occurrences : OccurrenceType[]

Reification: Kind implementations extends / plays Subject, Predicate and Object roles

in statement.

SubjectKind extends Subject, implements Kind<Subject, Predicate, Object>

PredicateKind extends Predicate, implements Kind<Predicate, Subject, Object>

ObjectKind extends Object, implements Kind<Object, Predicate, Subject>

The underlying model Statements can be represented as sets being Subjects,

Predicates and Objects three sets where the intersection of Predicates and Objects

sets conforms the “Subject Kinds” set, the intersection of the Subjects and Objects

sets conforms the “Predicate Kinds” set, the intersection of the Subjects and

Predicates sets conforms the “Object Kinds” set and the intersection of the three sets

conforms the “Statements” set. The set that encloses Subject, Predicate and Object sets is the Context set.

Sets based inference and functional algorithms should leverage this form of

representation of the model graph.

Statements:

Data: (Context, Subject, Predicate, Object)

Schema: (Context, SubjectKind, PredicateKind, ObjectKind)

**Dimensional Model (Reference Model view)**

Classes:

ContextStatement extends Statement(C, S, P, O)

Dimension

Attribute / Axis

Value / Measure

Statements:

(ContextStatement: recursive, Dimension, Attribute / Axis, Value / Measure)

Examples:​

(Time, soldDate, aProduct, aDate)​

((Time, soldDate, aProduct, aDate), Item, Product, aProduct)​

(((Time, soldDate, aProduct, aDate), Item, Product, aProduct), Region, Country, aCountry)

Encode inference of order relationships. Implement Order inference as a feature of the Dimensional Model: type (schema) and instances (data) hierarchies inferred in FCA Contexts.

**DOM Model (Reference Model view)**

Classes:

Instance extends IDOccurrence​

- id : ID​

- label : string​

- class : Class​

- attributes : Map<string, Instance>

Class extends Instance​

- id : ID​

- label : string​

- fields : Map<string, Class>

Statements:

(Class, Instance, Field, Instance);

**Activation (DCI, Actor / Role) Model (Reference Model view)**

Classes:

Context​

- roles : Role[]

Role extends Class​

- previous : Map<Context, Dataflow>​

- current : Map<Context, Dataflow>​

- next : Map<Context, Dataflow>

Dataflow extends Context​

- role : Role​

- rule : Rule

Interaction​

- actors : Actor[]

Rule: Dataflow specification.

Actor extends Instance​

- previous : Map<Context, Transform>​

- current : Map<Context, Transform>​

- next : Map<Context, Transform>

Transform​

- actor : Actor​

- production : Production

Production: Transform execution.

Statements:​

Data: (Context, Interaction, Actor, Transform)​

Schema: (Context, Context / Dataflow, Role, Dataflow)

**MESSAGES:**

Messages (Services and Models Statements exchange): Services / Components interactions and Registry Models storage is in the form of Reference Model Statements. Use Reference Model Statements as an implementation of an underlying model representation for Statement Messages exchange between Services and Models and persistence in the Model Service. Each service leverages the model type (view) most appropriate for its task.

**SERVICES:**

**Model Service**

Main RDF store persistence handler across services. Persistence of underlying Reference Model Statements and conversion back and forth between other model views handling. Keep core Reference Model state in sync with views models interactions and handle persistence within them.

**Datasource Service**

Datasource Service: consumes raw backends data, publish Statements (for Application Service). Listen for Statements (from Application Service), synchronizes backends data.

**Application Service**

Encloses Datasource, Augmentation, Producer Service interactions. Listen for Statements from Datasource Service / Producer Services. Dispatches to / consumes from Augmentation Service. Publish Statements to Datasource Service for backend sync / to Producer Service for Interactions state.

**Augmentation Service**

Encloses Aggregation, Alignment, Activation Service Interactions. Listen for Statements from Application Service. Dispatches Statements to / consumes from Aggregation, Alignment, Activation reactive functional pipeline. Publish Statements to Application Service.

**Aggregation Service**

Augmentation Pipeline step. FCA Prime IDs assignation, FCA Contexts creation / updates / sync. W3C DIDs assignation. Type / State / Contexts / Relationship / Order (hierarchies) inference. Consumes / Updates Model Service. CPPE / RCVs handling. Consumes / Updates Model Service. Alignment / sync of augmented input Statements with Model Service.

**Alignment Service**

[Alignment Service: Augmentation Pipeline step. Ontology matching. Upper ontology alignment. Link / Attribute prediction. Order inference (via Dimensional Model view). Consumes / Updates Model Service. Alignment / sync of augmented input Statements with Model Service.

**Activation Service**

Activation Service: Augmentation Pipeline step. Instantiates inferred use cases (DCI Activation model Contexts) executions (DCI Activation model Interactions). Enables use cases execution by means of Messages Statements exchange (state available verbs for actors transforms productions). Reference Model (corresponding views) alignment / sync of inferred use cases and their current executions. Implement COST / HAL Server implementation classes and a sample dynamic forms Angular client application. Show an example.

**Producer Service**

Producer Service: Listen for Statements (from Application Service). Publish Statements (to Application Service). Handles COST Conversations States.

**MCP Service**

MCP Server and MCP Client features in a separate service module. Enable connect Application Service as an MCP Client. MCP Server Tools: Activation Contexts, client retrieves use cases. MCP Server Prompt templates: Naming Service. Textual description (with placeholders) for Activation Interaction dataflow state transform step. Client retrieves interaction state possible prompts within that state. MCP Server Resources: Registry Service, client retrieve models representations from models service for Content Types. MCP Client features: Sampling, file system roots. Leverage Naming Service APIs.

**HELPER SERVICES:**

**Index Services**

FCA Contexts, Prime IDs, CPPE, RCVs Calculation. Similarity / Relationships Inference query / traversal API. Reference Model Inferences query / traversal API. Other models (views) Inference / query / traversal APIs.

**Naming Service**

Leverage LLMs / GenAI features (with Spring AI) for ontology matching, upper ontology alignment, links / attributes inference and order inference. Kinds, Relationships, Contexts, Roles name assignment / retrieval. MCP Server / Client features. Use CPPE / RCVs for embeddings.

**Registry Service**

Resource Repository View. Produce / Consume Resources in provided / requested Resource Content Types (Representations). Content Type encoders / decoders for Reference Model Resources APIs. Bi directional encoding / decoding APIs (mappings). Activation Interactions Actor transforms (Resource JAF Content Types activated verbs execution).

**FRONTEND:**

**COST / HAL WebUI**

WebUI (COST / HAL Generic Forms) Frontend. Producer WebUI: implement a reactive functional dynamic forms COST / HAL frontend in reactive Angular.

**ADMINISTRATION / CONFIGURATION:**

Reified Components / Services configuration data as Application Models Instances. Generate an Administration Services configuration schema (bootstrap) in the model, editable vía COST (Producer WebUI client) allowing to edit administration schema information from which Services / Components instances retrieve their configurations. Example schema (for Datasource Service: (datasource: (datasourceType, datasourceHost, datasourceUser, datasourcePass)). Implement this configuration mechanism for Services and Components as standard Spring Boot configuration methods / annotations. Service configuration should be treated as another case of backend integration whose configured datasource resides in an internal configuration persistence database and the activation interactions are performed at Service / Component configuration time setting parameter values as the interaction results values. Activation Context: Service configuration schema (roles: keys). Activation Interaction: Service configuration instance (actors: values). Provide a sample configuration dataset from which infer services configuration schema (initial integration Datasource example). Configure multiple types of datasources. Datasource writeBack example for each datasource type.

**CPPE / RCVs INFERENCE**

**TOPIC MAPS REFERENCE MODEL (RDF/XML / XSLT)**

**XSLT DRIVEN ACTIVATION TRANSFORMS**

**SEMANTIC OBJECT MAPPING**

**HOMOICONIC APPROACH (DATA AS CODE / CODE FROM DATA)**

Data, Information, Knowledge Model levels.

URN as a Resource (ID). Resource events examples: onCreation(Resource created), onOccurrence(Resource context), onAssociation(Resource association). TODO: define pipeline steps stream given components composition functions. Each pipeline Resource event triggers the main event loop pipeline stream from the beginning. Example: Naming (Aggregation) steps, Registry (Alignment) consumes and augments Aggregation Resources with its knowledge which in turn are published / consumed again for further Aggregation (type inference / embeddings / RCV schemas, for example). Idem for Index (Activation).

ID step augmented Resource can shield an Occurrence Resource event (onOccurrence). Occurrence step augmented Resource can shield an Statement Resource (onAssociation). Statement step augmented Resource can shield a Graph Resource (Flux<Statement>? Serialized Graph Representation?). Stream pipeline steps Functional Streams Composition Functions overloaded for each type of Resource: Resources Activation Content Type dispatch.

Functional Reactive Stream Pipeline Components:

Naming: URN Crafting / Matching. Aggregation. Functional Streams Composition Functions (Function<Resource, Resource> Resources).

Registry: Resource Repository. URNs Resolution / CRUD / Alignment. Functional Streams Composition Functions (Function<Resource, Resource> Resources).

Index: Resource Contents URNs Resolution (inferences, transforms). Activation. Functional Streams Composition Functions (Function<Resource, Resource> Resources).

Main Event Loop: Topic. Resources publishing / subscriptions. Functional Streams Composition Functions. onResource(Resource) : Resource. Naming, Registry, Index stream steps pipeline.

Reactive Resources: ID, IDOccurrence, Statement, Graph, Step, Messages, etc. Content Type Addressing: Graph Statements by patterns / CPPE / RCV schema instances / roles / Kinds / SPARQL. Statement occurrences by position / role (Kind). IDOccurrences by contexts (schema / Kinds). IDs by occurrences (role / context schemas). Content Types: graph/set, graph/activation, graph/reference, statement/activation, occurrence/subject, producer/form (COST state exchange), etc. Super type / sub type functional transforms (Resources Function<Consumes, Produces>): addressing / query / traversal / augmentation steps.

Resources: Data (XML), Transforms (XSLT). Addressable / Registered. Resolvable by Content Type / URNs pattern matching (Resource Messages Endpoint). Content Types: Inferred Schema Resources (RDFS). Upper Alignment. To / from underlying representation Transforms. Activation Verbs inference (Activation) inferred Transform Resources.

Pipelines: Step (Resource), Resource Messages Endpoint (Shared Messages Resource stream topic, Resources publish / subscribe. Obtain Resource representation in Content Type by URNs / pattern matching, retrieve / invoke activated Resource verbs), Shared State / Tools. Reactive Resources: ID, IDOccurrence, Statement, Graph, Step, Messages, etc. Underlying representation, Content Type renderers (Data out) / activation (Transforms out) registered handlers. Augmentation Aggregation, Alignment, Activation pipeline Resources.

Semantic Discoverable URNs (embed context / content). Reactive Resources subscribes / publishes to Messages Endpoint, Content Type / URNs pattern matching dispatches to corresponding event handlers resolving corresponding Transforms Content Types / URNs with this Resource as payload for each event. For updating FCA metadata / type, Kinds inference for example.

Transforms Resources applied over Content Type / URN matching Resources (Messages) payloads, publishes transformed results back to the Messages Resource.

Functional Composition: Main event loop. Pipeline Datasource Resource inputs, Messages Endpoint dispatch, Producer Resource outputs.

Custom Resources (IO Monad):

Datasources Resource Instances. Configured declaratively in the model. Produces / Consumes data URNs Statements (provenance / sync).

Producer Resource Instance. Produces APIs / UI Representation Content Types. Consumes user interaction forms state.

Wrap LLM / MCP into a Resource. Handle Datasources sync / Producer Activation COST.