**Models / Layers**

The idea is to enable model representations being equivalent (containing the same data) in various layers to be switched back an forth between each layer representation to be used in the most appropriate task for a given representation.

**Reference Model (Aggregation / Grammar)**

ID

- primeID : long

- urn : string

- occurrences : IDOccurrence[]

- embedding : double[]

IDOccurrence : ID

- occurringId : ID

- context : IDOccurrence

- embedding : double[]

Statement : IDOcurrence (Property Graphs)

- context : ID

- subject : ID

- predicate : ID

- object : ID

Statements:

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

Schema: (ID(IDOccurrence), ID(IDOccurrence), ID(IDOccurrence)

FCA Prime IDs (Embeddings):

Each ID is assigned a unique prime number ID at creation time. FCA Context / Lattices built upon, for example for a given Data / Schema predicate / arc occurrence role, having the context objects being the statement occurrence subjects and the context attributes the statement occurrence objects, Predicate FCA Context: (Subjects x Objects). For a subject statement occurrence the context is: Subject FCA Context: (Predicates x Objects and for an object statement occurrence role the context is: Object FCA Context (Subjectx x Predicates).

Embeddings: For an ID, its prime ID number plus all ID’s occurrences embeddings. For an IDOccurrence, its ID class embeddings, its occurring ID embeddings and its context embeddings.

Embeddings similarity: IDs, IDOccurrences sharing the same primes for their embeddings in a given context. FCA Concept Lattice Clustering. (TODO).

Statements:

(Context, Attribute, Value)

TODO:

FCA / Multidimensional features (OLAP like):

Dimensions: Time, Product, Region

Units: Month / Year, Category / Item, State / City

Context : (Context, Attribute, Value)

Examples:

(soldDate, aProduct, aDate)

((soldDate, aProduct, aDate), Product, aProduct)

(((soldDate, aProduct, aDate), Product, aProduct), Region, aRegion)

**Graph Model (Alignment, Semantics, Sets / Kinds)**

Context : IDOccurrence (Set)

Subject : IDOccurrence (Set)

Predicate : IDOccurrence (Set)

Object : IDOccurrence (Set)

Kind<AttributeType, ValueType> : ID / Interface

- superKind : Kind

- attributeValues : Tuple<AttributeType, ValueType>[]

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

SubjectKind : extends Subject, implements Kind<Predicate, Object> (Predicates intersection Objects)

- occurrences : Subject[]

PredicateKind : extends Predicate, implements Kind<Subject, Object> (Subjects intersection Objects)

- occurrences : Predicate[]

ObjectKind : extends Object, implements Kind<Predicate, Subject> (Predicates intersection Subjects)

- occurrences : Object[]

Statements:

Data: Context(Subject, Predicate, Object)

Schema: Context(SubjectKind, PredicateKind, ObjectKind)

[Sets Diagram]

**Activation Model (Activation, DOM / DCI / Actor, Role. Pragmatics)**

Instance : IDOccurrence

- id : ID

- label : string

- class : Class

- attributes : Map<string, Instance>

Class : Instance

- id : ID

- label : string

- fields : Map<string, Class>

Context

- roles : Role[]

Role : Class

- previous : Map<Context, Dataflow>

- current : Map<Context, Dataflow>

- next : Map<Context, Dataflow>

Dataflow : Context

- role : Role

- rule : Rule (TODO)

Interaction

- actors : Actor[]

Actor : Instance

- previous : Map<Context, Transform>

- current : Map<Context, Transform>

- next : Map<Context, Transform>

Transform

- actor : Actor

- production : Production (TODO)

COST (Conversational State Transfer) REST API is in initial state for a given context. The client retrieves the ‘current’ role context dataflow representation instance (Interaction, Actor, Transform), process it (DSL, ‘Activates’ and invokes API for the given representation Transform) and posts back the activated representation. The service then is able to determine the next Dataflow Role representation instance in a given use case (Context). TODO: Populate (infer) Dataflow Roles rules (state flows), Populate (infer / execute) Transform Actors productions using data encoded in the proposed models.

Statements:

Data: (Interaction, Actor, Transform)

Schema: (Context, Role, Dataflow)

**Helper Services:**

Index Service

- FCA / DIDs (ID Creation Time)

- Embeddings Index

- Store / index / query / update index and model’’s embeddings

Naming Service

- Resolve MCP interactions. Content type dispatch.

- Registry IO schema matching MCP resolution.

- Session interaction handler (MCP Resolves client resources / methods to invoke).

Registry Service:

- MCP Resources / contexts.

- Handle sessions / RAG (embeddings)

- IO Representations schema registry (MCPs Structured IO)

Storage Service:

- Internal Graph store (Neo4j / RDF4J)

- Message publish / dispatch. Model Layers events (schemas, content types, topics / queues routing).

- Consumer / Producer sync / bindings. Saga Pattern.

**Domain Services:**

Augmentation

Consumer

Aggregation

- FCA / Primes Contexts / Lattices.

- DIDs.

- Basic schema inference (IDOccurrences, IDs Statements input).

- Embeddings.

Alignment

- Type / State inference (Kinds)

- Align / Match Ontologies.

- Order / Dimensional features (Sets).

- Embeddings.

Activation

- DDD Populated Activation Model.

- Dynamic API Runtime (Conversational State Transfer: COST)

- Previous, Current, Next State Transfer IO from posting Index with Representations.

- Embeddings.

Producer

MCP (LLMs):

Define Reactive Streams Functional Processing APIs: MCP’s tools, prompts templates and resources). Content type dispatch. MCP Calling client methods / resources. Provider (Helper) Services: define APIs. Embeddings.

Tool (Reactive Microservice) interfaces (Functional APIs):

- inputs: consume streams messages matching pattern / content type.

- core tool logic: consume / produce from server. Callbacks

- produces: publish (registry) inputs / produces signatures. return / publish type structured outputs.