Application Service

Overview

Allow to integrate diverse existing / legacy applications or API services by parsing theirs backend's source data (in tabular, XML / JSON, graph, etc. forms) and, by means of aggregated inference using semantic models over sources schema and data, obtain a layered representation of the domains and data of source applications to be integrated until reaching enough knowledge as for being able to represent application's behaviors into an inferred use-cases Activation model.

Expose the Activation model inferred use-cases types (Contexts) and transactions use-cases instances (Interactions) through a Producer generic use-case browser client / API. Allow to browse and execute use-cases Contexts and Interactions in and between integrated applications, possibly enabling use cases involving more than one source integrated application. Example: Inventory integrated application and Orders integrated application interaction. When Inventory application level of one product falls below some threshold an Order needs to be fulfilled to replenish the Inventory with the products needed for operational levels.

The concept is to manage raw Datasources data and schema (inferred) into layers of Aggregation, Alignment and Activation services. Then the Producer component is able to parse and render Activation model into an application (API / generic frontend) Contexts and Interactions browser. An Augmentation service provides for orchestration between the three main layers of the service architecture and provides for interaction between Datasources and Producer services.

Models Architecture

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.

Reification: Statements could be about any type of URI (URIOcurrence(s)) in which Statements subjects, predicates and objects occurrences plays determinate role (Kind: Type / State) regarding this Statement occurrence context. Statements themselves are URIOccurrence(s) with their URIOccurrence uri being their subject URI, their statement being the statement itself (this) and their URIOccurrence Kind uri being their subject uri, their Kind type its predicate Kind Type and its Kind state being its object Kind State.

Those entities are to be able to be retrieved and their representations should enable functional programming techniques to be applied to streams of their representations to perform Aggregation, Alignment and Activation.

The nodes and arcs of the graph triples are URIs and should have a "retrievable" internal representation with metadata that each service / layer populates through the "helper" services: Registry, Naming (NLP) and Index service shared by each layer. Describe core model classes serialization in JSON.

Materialize. Reification of RDFS / OWL. Ontology Schema Statements. Same as. Schema (alignment) statements materialization.

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 Contexts. Prime IDs. Embeddings

FCA Prime IDs (Embeddings): (link Sowa)

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)

URIs are identifiers (Strings) and have assigned an unique prime number ID at their creation time. FCA (Formal Concept Analysis) techniques could be employed to build a concept lattice for each URI in a given context where the product of the primes of the URI context occurrence concept lattice attributes and values URIs are employed to identify the concept the URI belongs to and to subsume other possible attributes.

Graph Model

(Alignment, Semantics, Sets / Kinds)

Context : IDOccurrence (Set)

Subject : IDOccurrence (Set)

Predicate : IDOccurrence (Set)

Object : IDOccurrence (Set)

Kind<AttributeType, ValueType> : Interface
- superKind : Kind

- attributeValues : Tuple<AttributeType, ValueType>[]

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

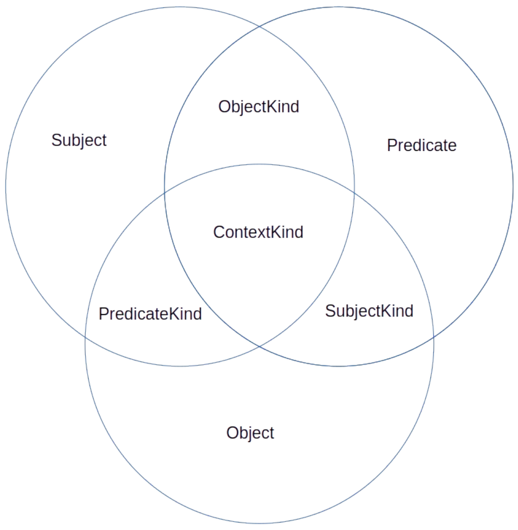
SubjectKind : extends Subject, implements Kind<Predicate, Object> (Predicates intersection Objects)
- occurrences : Subject[]

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

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

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.

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)

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)

Statements

Data: (Interaction, Actor, Transform)
Schema: (Context, Role, Dataflow)

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.

The goal is to integrate the domains and functionality of various applications into a unified and integrated API or interface (unified front end). Given all the application / services to integrate: Extract all data sources from the applications to be integrated and represent them in a unified way. Perform Augmentation (Aggregation, Alignment and Activation) over the source raw data and schema to achieve an unified interface exposed through an unified API Consumer Service which exposes the Contexts (Use Cases) and Interactions (Use Case executions) inferred and possible in and between integrated applications (REST API).

Components (Services) Architecture

The idea is that by doing an "ETL" of all the tables / schemas / APIs / documents of your domains and their applications, translating the sources into triples (nodes, arcs: knowledge graph) the framework can infer your entity types, relationships and the contexts ("use cases") possible in and between your integrated applications providing means for a generic overlay (Producer API Service, generic front end) in which to integrate in a unified, conversational and "discoverable" interface (API, web assistant, “wizards”) the integrated contexts interactions in and between the source integrated applications.

To unify and integrate diverse data sources, transform all the information from each source into triples (Entity, Attribute, Value) into a graph in the "Datasources" component. The other components / services deal with type / state inference (Aggregation), relationships and equivalences / matching / ordering (dimensional) inference (Alignment) and use case descriptions / executions (Activation) then exposing the description of the possible contexts and their interactions in and between the integrated applications. The user interface component could be a generic front end or an API endpoint to interact according to the metadata of each context (use case) augmentation allowing to make possible Contexts executable and their executions (Interactions) browseable.

Simple example (use cases): I have fruits and vegetables, I can open a greengrocer's. I want to open a greengrocer's, I need fruits and vegetables. Actors: supplier, greengrocer, customer. Contexts / Interactions: supply, sale, etc.

Another example: I have these indicators that I inferred from the ETL, what reports can I put together? I want a report about these aspects of this topic, what indicators (roles) do I need to add.

Ultimately, it is about creating a "generator" of unified interfaces for the integration of current or legacy applications or data sources (DBs, APIs, documents, etc.) in order to expose diverse sources in an unified way, such as a web frontend (generic use case wizards), chatbots, API endpoints, etc. integrating the functionality of integrated applications use cases relating each other in an unified forms flow layout (wizards).

Services Layout

Services: Reactive Endpoints. Consumers / Producers <MessageType>. WebFlux: onMessage (post), nextMessage (subscribe). Reactive Consumers / Producers with Message IO in context (session / dialog history) #EAP.

All services should have an administration / management interface for each step of the workflow. Example: Add datasources, view inferred types and their instances, view aligned upper ontologies (endpoint), view current contexts / interactions, browse available API endpoints definitions.

The communcation between services is in the form of serialized core model statements messages and events which each service process and augments in a functional reactive manner a core model graph in the helper Registry Service, performs upper ontologies alignment and matching in the helper Naming Service and provides for a repository of aligned resources to be activated (created, retrieved and updated) in the helper Index Service.

Application Service

Datasource Service

ETL / Synchronization with the backend datasources of the integrated applications providing and consuming graph models streams handling provenance and comsumption / updating of the integrated applications backend datasources.

One basic translation from tabular data could be to represent a row in a source application database as a triple in the form: (S: Row PK value, P: Row Column Name, O: Row Column Value) for a given PK value).

Inputs / Outputs: Core Model Classes Statements (see below). SPO Triples.

Communication with the Datasource Service, integrated applications data retrieval and synchronization / update (provenance) is between the Datasource Service and the Augmentation Service. Augmentation Service dispatch messages and events with context between the Datasource Service and the orchestrated services via Helper Services messages and events, retrieving the needed information and providing Datasource Service with updated information.

Augmentation Service

Handles Services Subscriptions. Dispatch Kafka (Reactor) Streams. Saga Pattern (MessageType logs). Reactive Consumer / Producer with Message IO in context (session / dialog history) #EAP.

Orchestrates core services (Aggregation, Alignment, Activation) feeding the Aggregation service with the graph models streams provided from the (synchronized) Datasources Service and provides the Consumer API Service with the Activation streams facilities to instantiate Contexts (use cases) and perform Interactions (transactions).

Provides helper (orthogonal) services access to the orchestrated services (Aggregation, Alignment and Activation) which enable for the functional (streams) manipulation of the Core Model Classes Statements (see below) between services.

Aggregation Service

Consumes: DataSources (CSPO) URI Strings Statements.
Produces: Reference Model (ID, ID, ID, ID) Statements.
Features:

- IDs / Embeddings.
- FCA Contexts (Clustering).
- Basic types / attributes inference (FCA).

Given a set of raw SPO triples from Datasources Service, performs type inference (common attributes aggregation) and state inference (common attribute values aggregation) and performs type / state hierarchies inference.

Type inference: Subjects with the same Attributes belong to the same type.
State inference: Subjects with Attributes (types) with the same Values are in the same state.

Type / State hierarchies:

Entities with the same attributes are considered as of the same type, superset / subset of attributes: type hierarchy. Attributes with the same values, same states. Superset / subset of values / states: state hierarchy.

Types are ordered in respect to their common attributes. Most specific types (more common attributes) are considered to inherit from types with less common attributes included into the more specific types. A more specific type is considered to be “after” a more generic type (Person → Employee). Regarding state values, hierarchies are to be considered regarding attribute values, being resources with common state grouped into hierarchies (Marital status attribute: Single → Married → Divorced).

Order: Inferred via Type / State hierarchies. Types: Married extends from Single, Divorced extends from Married. States: Young extends from Child, Old extends from Young. Cycles in types resolved by state (Unemployed, Employed, Unemployed). Used in (2.4) Alignment Service Ordering upper ontology.

Map<Subject, Set<Predicate>
Map<Set<Predicate>, Type>

Map<Type, Set<Map<Predicate,Value>>>
Map<Set<Map<Predicate,Value>>, State>

Inputs / Outputs: Core Model Classes Statements (see below). Leverages ML Classification.

Alignment Service

Consumes: Reference Model (ID, ID, ID, ID) Statements.
Produces: Graph Model (CSPO) Statements.
Features:

- Upper (inferred) ontology alignment.
- Links completion, Ontology Matching.
- Types / Kinds Inference. Order (hierarchies / dimensional).

Aligns (links / attributes, ontology matching, upper ontologies alignment) Aggregation Statements. Augments overall model.

Upper ontologies:

a) Domains: Aligned integrated application domains inferred common concepts and relationships. Infer equivalent concepts and relationships between source applications domains and populate Domains upper ontology. Materialize integrated domains concepts and relationships mappings to inferred upper concepts and relationships. Abstract common meaning (semantics) of source applications concepts and relationships to enable inter domain contexts interactions.

b) Order: Dimensional arrangement of entities attributes and values. Align measures (attribute values) into dimensional units. According Aggregation Service types and states hierarchies establish order relationships (before, greater than, contains, etc.) between measures. Materialize measures relationships and map dimensional units measures occurrences into the materialized order relationships. See: [5. Dimensional Features].

Ontology Matching: Find and map equivalent entities and relationships domains occurrences (Core Model Classes), align core model resources into Domains upper ontology.

Links / Attributes inference: Given an aligned model (mapped to Domains upper ontology) infer possible links / relationships between resources and possible attributes and their values.

Ordering: Order dimensional upper ontology alignment. Type / State hierarchies

Inputs / Outputs: Core Model Classes Statements (see below). Leverage ML Clustering.

Activation Service

Consumes: Graph Model (CSPO) Statements.
Produces: Activation Model DCI (Context, Interaction, Role, Actor) Statements.
Features: Use case inference / execution. Browse / Run transactions across integrated applications.

Activates Resources discovering from their types, states and order relationships which Use Cases (Contexts) are available in and between Resource types, states and order and which Roles are played by which types in state and order and allows to instantiate Transactions (Use Case Contexts Interactions) assigning Actors Resources to play specific Context Use Case Roles. The business logic of each Transaction (data flow) between Actors of different integrated domains applications playing Roles in a Context Interaction is to be inferred from the Alignment Service upper ontologies (Domains and Order).

Contexts, Roles, Interactions and Actors are inferred and aligned to an Activation upper ontology leveraging Alignment Service Domains and Order upper ontologies.

Activation upper ontology should enable Consumer API Service to expose available Contexts, Contexts state (Interactions instances), instantiate Contexts into new Interactions and fulfill Interactions Context Roles with the playing actors for this transaction and performs any steps involved in the creation of the current transaction (steps, forms flow, wizard like interface).

Activation upper ontology should be able to be queried by the Consumer API Service to build an Context Interaction scenario given a desired Context Interaction transaction outcome, letting the Activation Service populate possible Context Roles Actors for the desired outcome and showing possible scenarios to the user. Activation upper ontology follows the guidelines of the DCI: Data, Context and Interactions design pattern, letting the part of the transactions ordered steps / invocations data flow to be inferred from the current aligned Context Interactions transactions instances materialized in a declarative fashion into the model.

Data flow encoding:

(Contexts / Roles, Interactions, Actors) : Kinds(Type, State).

(Buy, Product, Good
(Good, Price, Amount)
(aBuy, contextType, Buy) : has ContextType → Interaction
(aBuy, Product, aProduct
(aProduct, Price, anAmount)

(anAmount, buyer → seller); (aProduct, seller → buyer);

Infer / Materialize / Perform operations. Encode functional mappings: assign / transform roles attributes.

Inputs / Outputs: Core Model Classes Statements (see below). Leverages ML Regression.

Contexts, Roles / Interactions, Actors
Contexts Actions flows and actions behavior declaratively stated from inference into dynamically stated logic / dataflow (XSLT Transforms generated from inference) into flows of reactive streams. SPARQL Backend CRUD, MCP Tools / Server.

Producer Service

Communication with the Producer API Service, unified REST APIs exposure, is between the Consumer API Service and the Augmentatio Service. Augmentation Service dispatch messages and events between the Consumer API Service and the orchestrated services via Helper Services messages and events, retrieving the needed information and providing

Consumer API Service with updated information (dialog conversational state).

Find relationships and equivalences between the data of the applications to be unified and their possible interactions. Use cases in and between applications.

Expose through an API the possible interactions to be invoked, their contexts roles and transactions interactions actors, and synchronize transaction data with the original applications. Provide a generic API Service front end (REST / Web). Provide a generic forms front end for rendering Contexts Interactions instances.

One should be able to ask for Contexts Interactions with a desired outcome, via inference performed determining which Actors should play which Roles in which Interactions (state, order) to achieve which Context Interactions results are desired.

Example: Launch new product to the market Context. Manufacturing, Inventory, Orders Delivery and Public Advertising integrated applications interacts as Actors with their respective roles in each step of the Launch new product to the market use case (Context) instance (Interaction).

One should be able to navigate previous Interactions (Contexts executions) or to create new ones (Contexts invocation).

Generic REST API Frontend: Exposes Activation Service Contexts Use Cases and allows to create, browse, update or continue existing Contexts Interactions transactions.

Possible Scenarios: Given a desired outcome, browse possible actors in context roles that would fulfill the desired result.

Gestures (Functions. Content Type available verbs). Domain Driven Design.

Forms / Flows: Roles Placeholders, Actor Values given Context. HATEOAS / HAL.

Inputs / Outputs: Core Model Classes Statements (see below). Leverages ML Regression.

Services Dataflow (see Protocols)

DataSource ↔️ ApplicationService ↔️ Augmentation ↔️ Aggregation ↔️ Alignment ↔️ Activation ↔️ Augmentation ↔️ ApplicationService ↔️ Producer

Helper Services

MCP Host. Servers. Structured Outputs / Inputs (prompt templates).
DIDs / FCA Contexts / PrimeIDs.
Index / Embeddings / Similarity.
Shared State (representations aware backend).
Message Types Marshall / UnMarshall (models / layers interoperability).
#Tools

Index Service

TODO

Repository of aligned resources to be activated (created, retrieved and updated) in the Activation service via similarity resolution. Dialog state based interface (Conversational State Transfer).

Resolve possible / actual contexts / interactions given resource representations. Resolve interaction possible / populated context templates (actors in roles placeholders).

Given a Resource representation in a given context and a given verb (Content Type method), retrieve the next Resource representation in the Activation flow (form with Content Type placeholders). Consumer fills in forms placeholders and the index is asked to retrieve again the next Resource representation for a given verb in the Activation flow.

Streams / events based interfaces.

Naming Service

TODO

Upper ontologies (Domains, Dimensional Order and Activation) matching and alignment.

Sets (See: [4 Sets Representation]) internal inference model representation. Sets API and functional set processing operations for matching and alignment tasks.

FCA (Formal Concept Analysis) contexts representation. Concepts Lattices for concepts alignment and attribute inference.

Links / relationships resolution in contexts. Attribute values, Context interaction roles. Order inference materialization.

SPARQL Endpoint. URI Based retrieval. Streams / events based interfaces.

Registry Service

TODO

Core graph model repository. To store / retrieve / share results of streams functional processing in each Augmentation (Aggregation, Alignment and Activation) orchestrated services. Hierarchical key / value store. TMRM (ISO Topic Maps Reference Model). Provenance repository (applications datasources synchronization). Embeddings.

SPARQL Endpoint. URI Based retrieval. Streams / events based interfaces.

Protocol (Message Types) Architecture

Context / History (session) aware Dialogs #EAP

TODO

CSPO URI Strings Statements

Statement<String, String, String, String>.

Reference Model Statements

Statement<ID, ID, ID, ID>.

Graph (Sets) CSPO Statements

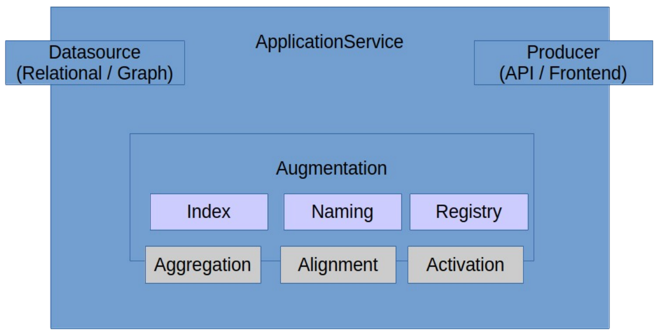
Statement<Context, Subject, Predicate, Object>.

Activation (DCI) Statements

Statement<Context, Interaction, Role, Actor>.

Interactions (Reactive Services Messages Dataflow) Architecture

Functional (Use Case) Architecture



Integration Example

Configured Datasources of Applications to integrate from (examples). Produces (CSPO) URI Strings Statements Streams.
Aggregation: ETL Consumes previous Statements, Produces Reference Model IDs / IDOccurrences Statements Streams.

Prime IDs / DIDs Assignation. DCI Contexts Concept Lattices Tables population.

Alignment matches sources equivalences.

Activation discovers available Contexts Interactions and past Interactions Transactions.

Producer UI selects next step from current state (available transaction or next if previous state). Submits Context Interaction Role Actor Form: Browses previous or perform next.

Activation consumes Producer statements and forwards to Alignment for backend sync processing.

Alignment matches Activation inferred Contexts into DataSources source data.

Aggregation converts Alignment source aligned data into Datasource quads.

Augmentation submits stream for updating and syncing to Datasource.

TODO

Dimensional Features

Dimensional Upper Ontology.

Type / State hierarchies.

Entities with the same attributes are considered as of the same type, superset / subset of attributes: type hierarchy. Attributes with the same values, same states. Superset / subset of values / states: state hierarchy.

Types are ordered in respect to their common attributes. Most specific types (more common attributes) are considered to inherit from types with less common attributes. A more specific type is considered to be “after” a more generic type (Person → Employee). Regarding state values, hierarchies are to be considered regarding attribute values, being resources with common state grouped into hierarchies (Marital status attribute: Single → Married → Divorced).

Order encoding (octal).

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.

Semiotics: Context / Sign, Role / Object (SPO). Recursive (parts / whole).

Data / Information / Knowledge Services Layers separation:

Data: (Aggregation Statements)
Type (Attributes) / State (Attribute Values)
Example: Product price.

Information: (Alignment Statements

Matching / Linking (Domains upper ontology alignment) / Ordering (Dimensional upper ontology alignment)
Example: Product price variation.

Knowledge: (Activation Statements)
Contexts Roles
Interactions Actors
Example: Product price tendency (increase / decrease) over time (ordered price values variation across time dimension).

Dimensional Relationship Statements:
Measures: (Dimension, Unit, Value)

Speed Measure: (Speed, “Kilometers per hour”, 120)
Distance Measure: (Distance, “Kilometers”, 120)
Time Measure: (Time, “Hours”, 1)

Translations:
Speed / Time: (speed, distance, time);
Speed / Distance: (speed, time, distance);
Distance / Time: (distance, Speed, Time);
Distance / Speed: (distance, Time, Speed);
Time / Speed: (time, distance, speed);
Time / Distance: (time, speed, distance);

Features to explore:

There is something called "Web3" that uses decentralized blockchain for the management of identifiers (URIs as DIDs: W3C Decentralized Identifiers*) and their interactions and semantics (smart contracts for example). Since the nodes and arcs of the graphs are URIs, it would not be unreasonable to use the Java APIs that are available on GitHub for this (DIDs) to facilitate the interaction of different instances or deployments of this framework between different organizations.

[Explain W3C DIDs Use Cases in the microservices architecture]

The following is a spare list of topics / keywords which should be considered regarding implementation features and related tools that could be used during implementation:

Semantically Annotated Hypermedia Resources / Objects Addressing. HyTime / XML. ISO Topic Maps / ISO 15926. W3C RDF. Addressable Hypermedia / Hypermedia

Addressing Augmentation and linking (actors, roles and contexts interactions).

TMRM (Topic Maps Reference Model) / TMDM (Topic Maps Data Model) like SPO URIs underlying representation embeddings.

Representation / Functional Transforms: XML / Dynamic XSLT (codat). De referenceable Resources Representations (functional layers 'views'). Reactive Functional Engine (service layers streams XML / XSLT).

Semiotic Layer: Objects / Signs Concepts Occurrences in Contexts. Hypermedia Augmentation / Annotation: Aggregation, Alignment, Activation Functional Definitions (domain / range). Occurrences.

Layers inputs / outputs. Designer (Service layer management interface).

Streams Flow. Layers Functions.

Function<URI / Resource, URI / Resource>(Statement[] stats / URI strategy). Functional Monadic Parser.

Functional "getters" / "setters" (Monads traversal).

Association rule mining. Regression.

Activation: Resource Content Type Capabilities.
◦ Buy-able (Transaction, Product)
◦ Identify-able (Features, Image)
◦ Locatable (Space, Position)

Domains. Alignment. Upper Ontologies.

Encoding (definitions / assertions: rules / grammar / productions).

TMDM / TMRM, RDF / OWL (ISO).

Encoding: Naming, Index, Registry. Context / Roles Definitions, Interactions / Actors Assertions. Apply Functional Transforms.

Encoding: Semantic Virtual Machine.

Representation Levels. Transforms. Context / Sign / Concept / Instance level operations (dataflow / transforms).

Drop-able (drivers) ML Models (Activation). LLM Outlined (Naming, Index, Registry) functional abstractions integration backed (MCP Activation / streams contexts resolution).

Core DOM. Type Object / Actor Role Pattern Implementatíon. DCI. Qi4j. DDD.

Upper / Domain (inferred / aligned) DCI Use Cases ontologies alignment. Fine grained (operations / dataflow). Coarse grained (transactions).

LLMs / MCP / Agents / ML Foundation APIs. SCDF Like, tools / streams / events bindings. Functional Services / Workflows. GraphNNs. Ontology Alignment / Matching.

Lectures / Bookmarks. Syllabus.Summarize (outline): Features, Lectures / Bookmarks refs. Spring AI + MCP + Reactive Functional Streams + FCA (embeddings), DCI, etc. Encodings (embeddings). Inference server. Ollama.

Semantic Virtual Machine (embeddings). State transitions dataflow / behaviors schema / instances (previous, current, next). Embeddings (IndexService, Blockchain DIDs). Semantic Virtual Machine: Graph traversal layout arrangements. Schema / instance embeddings possible states flow (previous , current, next) dataflow given (other flows traversal) context. Possible graph transitions.

Tools