**1. Overview**

Reactive microservices framework which allows to integrate diverse existing applications / services / backends and the use cases and interactions possible in / between them by means of using Semantic Web technologies and inference leveraging reactive functional programming streams processing. Parse integrated applications source data and, by means of inference and learning, provide an unified interface to browse / perform possible use cases and their interactions (transactions) for each integrated application and possibly use cases involving more than one application in a transaction. Example: Inventory integrated application and Orders integrated application interact when Inventory application level of one product falls below some threshold and an Order needs to be fulfilled to replenish the Inventory with the products needed for operational levels.

The concept is to manage raw data (SPO Triples) inputs into layers which perform Aggregation (Type, State, Order inference), Alignment (Attribute / Values Relationships inference, upper ontology alignment and ontology matching) and Activation (Roles / Actors, Contexts, Interactions inference, DCI: Data, Context, Interaction design pattern) given an enclosing orchestrating Augmentation Service which hooks datasources (Datasource service) inputs with consuming API Service / Generic Frontend Service (Generic discoverable / browseable Activation Use Cases (Contexts / Interactions) APIs from Activation Service metadata).

**1.1 Features**

RDF SPO Graphs from tabular sources / others: to distill the metadata needed for Aggregation, Alignment and Activation, tabular sources and other kinds of persistence methods are to be translated into RDF Triples. One such basic translation could be to represent a row in a database as a triple in the form: (S: Row PK value, P: Row Column Name, O: Row Column Value for a given PK value).

Once Aggregation, Alignment and Activation has been performed one should be able to manipulate Datasources Data at Context (use case) level, with Actors (data) playing Roles (Types) in interactions (Contexts execution interaction instances from the consuming API Service).

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).

**1.2 Business Integration and EAI through Semantic Web**

This is a first draft of a document in respect to what could be a BI (Business Integration) and EAI (Enterprise Applications Integration) through Semantic Web framework / toolkit.

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).

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.

The idea is that by doing an "ETL" of all the tables / schemas / APIs / documents of your domain and 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 generating a generic overlay (Consumer 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.

**2. Services Layers / Components:**

**2.1 Augmentation Service:**

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.

**2.2 Datasources 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.

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

**2.3 Aggregation Service:**

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. ML Classification enabled.

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).

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).

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

**2.4 Alignment Service:**

Links / Attributes inference. Order / Upper ontology alignment. ML Clustering enabled.

Ontology Matching. Find and merge equivalent entities and relationships (Core Model Classes). Align core model resources into an upper ontology.

Dimensional upper ontology alignment (order). See below.

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

**2.5 Activation Service:**

Actors Activation in Contexts Interactions Roles inference according Aggregation and Alignment Metadata. ML Regression enabled.

Contexts (Use Cases) with their Roles and Contexts Interactions (Contexts executions) are encoded into the core model classes and aligned into an activation upper ontology exposing DCI (Data, Context and Interactions design pattern) metadata needed for the Consumer API Service for building API services.

Use Case Types (Contexts / Roles) and Instances (Interactions / Actors) inference. APIs description metadata. DCI (Data, Context, Interaction) Design Pattern. Browseable possible / past interactions (transactions) for each context / actors roles.

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

**2.6 Consumer API Service:**

Generic REST API Frontend: Generic discoverable / browseable Use Case (Contexts / Interactions) APIs from Activation Service metadata. Possible / past interactions (transactions) REST API. Gestures (Functions. Content Type available verbs). Domain Driven Design. Context: traversal state, referrer resource. Instances: Resource occurrences. Infer Templates (Forms), Roles / Placeholders / Values given Context. HATEOAS / HAL.

**Overall services layout:**

All services are orchestrated through the central / enclosing Augmentation Service.

All services should have an administration / management interface for each step of the workflow with a graph-oriented backend, leveraging Graph NNs and LLMs / NLP through functional / reactive programming (streams) of the orchestrating Augmentation (flow) microservice and their tasks.

Layers reactive streams flow:

DatasourceService ↔ AugmentationService ↔ ConsumerAPIService

AugmentationService ↔ AggregationService

AugmentationService ↔ AlignmentService

AugmentationService ↔ ActivationService

Each service layer consumes and produces streams (serialized core model classes in a reactive functional programming fashion) from the previous and following service layer respectively performing functional transforms (over monadic core classes representations) retrieving and updating Entities (core model classes) data and metadata leveraging helper services (Registry, Naming, Index).

**2.7 Helper Services (shared via AugmentationService):**

**2.7.1 Registry Service:**

URI Based repository (CRUD) of all URI identifiable / retrievable concepts (streams) entities. To store / retrieve results of streams functional processing. Aggregation (Type / State / Order) hierarchies aware functionality. Hierarchical key / value store: TMRM (ISO Topic Maps Reference Model) LHS, RHS encoding. Provenance (applications sync). ML / LLM Enabled.

**2.7.2 Naming Service:**

Resolve equivalent / matching identifiers in contexts. Links / relationships resolution (roles / names). Upper ontology alignment. To retrieve equivalent entities / relationships in contexts. Alignment aware functionality (streams processing / functional programming). Grammar. Rules. NLP / NER. Mappings (Templates / Placeholders. Forms). ML / LLM Enabled.

**2.7.3 Index Service:**

Similarity entities in contexts resolution. Resolve possible / actual contexts / interactions given Aggregation entities in Alignment contexts. Resolve interaction possible / populated context templates (actors in roles placeholders). Embeddings. ML / LLM Enabled.

**3. Core Model Classes:**

The core entity concepts / classes to be “streamed” between functional layers could be the following:

URI {

resource : String

primeId : long

occurrences : URIOccurrence[]

}

URIOccurrence {

uri : URI

statement : Statement

kind : Kind

}

Resource extends URIOccurrence {

contentType : String

entityBody : String

}

Type extends URIOccurrence {

supertype : Type

}

State extends URIOccurrence {

superstate : State

}

Kind extends URIOccurrence {

type : Type

state : State

superkind : Kind

}

Statement extends URIOccurrence {

subject : URIOccurrence

predicate : URIOccurrence

object : URIOccurrence

}

Statement as URIOccurrence uri : subjectURI

Statement as URIOccurrence statement : this

Statement as URIOccurrence kind : Kind(uri : Statement subject URI, type : Statement predicate Kind type, state : Statement object Kind state)

[Class Diagram Here]

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.

**3.1 FCA (Formal Concept Analysis)**

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.

**4 Sets Representation:**

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.

[Sets Diagram Here]

**5. Dimensional Features (Helper Services):**

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).

Business Intelligence Data / Information / Knowledge separation:

Data: (Aggregation Triples SPO)

* + Type (Attributes)
  + State (Attribute Values)
  + Order (By means of Type / State hierarchies)
  + Example: Product price.

Information: (Alignment Triples vía Metadata Type, State hierarchies relationships)

* + Information (Type / State relationships)
  + Linking (Entity relationships). Ordering (upper ontology alignment)
  + Equivalence (Entity, Type, State matching)
  + Example: Product price variation.

Knowledge: (Actors Activation in Contexts Roles Interactions given available Information and Data)

* + Contexts
  + Roles / Actors
  + Interactions
  + 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)