To develop a 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).

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).

**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: PK value, P: Column Name, O: Column Value for a given PK row).

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).

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

For example, in the 'Family' context:

A cousin is the son of the brother of the father of another son.

An interaction could be:

Peter is the son of John.

John is the brother of Arthur.

Arthur is the father of Charlie.

Peter is the cousin of Charlie.

**Aggregation**

Inputs: RDF: W3C Resource Description Framework SPO Triples.

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.

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).

**Alignment**

Inputs: Aggregation Triples augmented with Type, State and Order Metadata.

Information inference: Attributes / Values Relationships.

Linking inference: Entities Relationships. Upper ontology alignment.

Equivalences inference: Entities, Attributes, Values Ontology Matching.

**Activation**

Actors Activation in Contexts Interactions Roles according Aggregation and Alignment Metadata.

Inputs: Aggregation and Alignment augmented Triples.

Contexts inference.

Interactions inference.

Contexts Roles / Interactions Actors inference.

**BI 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:

Integrate the domains and functionality of various applications into a unified and integrated API or interface (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 Service which exposes the Contexts (Use Cases) and Interactions (Use Case executions) inferred and possible in and between applications.

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).

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 between your applications generating a generic overlay (API Service, Frontend) in which to integrate in a unified, conversational and "discoverable" interface (API, web assistant, “wizards”) the integrated contexts interaction in / between the source integrated applications.

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

The architecture would be microservices with five components, which for now are "black boxes", interfaces for reactive microservices to implement their algorithms with functional / reactive streams programming.

The components are:

\* Datasources Service: ETL (tabular, APIs, documents to RDF SPO triples knowledge graph). Populate initial graph. Synchronization with the backends of the integrated source applications according to the interactions of the contexts (Activation inference generated APIs).

\* Aggregation Service: Contexts / Relationships, Types, States inference. From the "raw" data, infer types and meta-types (state) of the entities of the datasources to be integrated through their attributes and their values in a given context (relationships). ML Method used: Classification.

I consider entities with the same attributes as the same type, superset / subset of attributes: type hierarchy. Attributes with the same values, same states. Superset / subset of values / states: state hierarchy. Given hierarchies: order inference (as stated above).

\* Alignment Service: Ontology Matching. Find equivalent contexts / types / states / entities / relationships. Missing Links / Attributes inference. Upper ontology alignment. ML Method used: Clustering.

\* Activation Service: 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. ML Method used: Regression (Contexts, Roles, Interactions, Actors).

\* Consumer API Service / Generic 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.

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

All services would have an administration interface for each step of the workflow with a graph-oriented backend (RDF4J or Neo4j), leveraging Graph NNs and LLMs / NLP through functional / reactive programming (streams) of the composing Augmentation (flow) microservices and their tasks.

TODO: Define the "schema" of the graphs for each input / output of each component. Through functional and "reactive" programming, implementing algorithms that incrementally "parse" graphs and their respective inferences in each service so that the system is dynamic and iterative (incremental integration). Handle synchronization between services layers: An API Service invoking Activation service contexts into interactions, interactions should update integrated applications datasources (Datasource service).

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.

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.

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.

The services architecture could be depicted as this:

DatasourceService (data retrieval and backend synchronization)

AugmentationService (enclosing orchestration service)

AggregationService (type / state / order inference)

AlignmentService (link / equivalence matching, upper alignment)

ActivationService (use case contexts level layer, contexts interactions instances)

Consumer API Service (REST API for contexts / interactions management)

Each service layer consumes and produces streams (reactive functional programming) from the previous and following service layer respectively.

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

URI

URIOccurrence (uri : URI, statement : Statement, kind : Kind)

Entity / Resource (uri : URI) : URI

Type (uri : URI, supertype : Type) : URI

State (uri : URI, superstate : State) : URI

Kind (uri : URI, type : Type, state : State) : URI

Statement (subject : URIOccurrence, predicate : URIOccurrence, object : URIOccurrence) : URIOccurrence(uri : subjectURI, statement : this, kind : Kind(uri : subjectURI, type : predicateType, state : objectState)

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.

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 Type and its Kind state being its object occurrence state.

Helper Services (orthogonal to the whole Augmentation services):

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

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 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.

Order encoding (octal):

TODO

Helper Services (orthogonal to other services):

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.

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.

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.

Frameworks to be used:

In principle this would be a Spring Microservices application with Spring Reactive Extensions (Rx API) enabled and related tools (Spring Data for example) from the Spring ecosystem.

---

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.

Sets: 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.

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.

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).

Activation: Resource Content Type Capabilities.

◦ Buy-able (Transaction, Product)

◦ Identify-able (Features, Image)

◦ Locatable (Space, Position)

Aggregation: Classification determines content types. Inputs / Outputs.

Alignment: Clustering / Detection (parts / occurrences classification / roles) determines relationships / equivalences (inferred upper ontology / same as). Inputs / Outputs.

Activation: Regression determines contexts roles interactions actors states (discrete: class / roles, continuous: values) in facets (respect to other actor / roles actions, e.g.: buy / sell price). Inputs / Outputs.

Dimensional Relationship Statements:

• Speed: (anSpeed, distance, time);

• Speed: (anSpeed, time, distance);

• Distance: (aDistance, Speed, Time);

• Distance: (aDistance, Time, Speed);

• Time: (aTime, distance, speed);

• Time: (aTime, speed, distance);

Context Inference examples:

* A cousin is the son of the brother of the father of another son.
* A potential buyer of this new product is like…
* A potential new product for this audience has the following characteristics...
* Transaction X had place between Y and Z for a product P for an amount S.
* Y: Has Product. Z: Has amount. Interaction Roles: X.buyer, X.seller, X.P.price, etc.

Business Intelligence Data / Information / Knowledge separation:

* Data: (Aggregation Triples SPO)
  + Type (Attributes)
  + State (Attribute Values)
  + Order (By means of Type / State hierarchies)
* Information: (Alignment Triples vía Metadata Type, State and Order relationships)
  + Information (Type / State relationships)
  + Linking (Entity relationships)
  + Equivalence (Entity, Type, State matching)
* Knowledge: (Actors Activation in Contexts Roles Interactions given available Information and Data)
  + Contexts
  + Roles / Actors
  + Interactions