## Application:

Products And Services Community Exchange Network:

Contents / Features (Mision / Vision). Distributed consistent Knowledge Applications. Trust. Consistency. Event sourcing. Inferencing (of distributed state). Reconciliation.

Certify distributed Entity / Subject Identity / State (in roles / dimensional points). Class / instance alignment (matching).

Integration: Augment sources / back ends. Model I/O materialized in source (plugged) application / services back ends.

Integration: Extension. Extended functionalities data / schema / behavior exposed as services external to source (plugged) applications. Sync (Augment). Declaratively stated via Model descriptions. Discoverable, browseable (HAL / REST).

The idea of the project is to "augment" an ESB for EAI platform and to enable it allowing it to make "inferences" regarding which routes to use, "discovering" sources / destinations of an event message(s) which then it transforms / enriches according destination "semantics" and format(s).

This featuring the exposure of a generic facade which allows to see in an "homologated" view the applications or services and their data, schema and behavior (actions) that could be integrated into the tool.

Different integrated applications are enriched with this facade and with the events that, given the inferred routes and transformations, augments theirs data, schema and behaviors, invoking activities corresponding to each destiny semantics.

Going through my most recent attempts of having something concrete for sharing in plain English I realize one mistake I'm committing: I'm trying to describe combustion vehicles (Hypermedia Applications) saying that petroleum exists (Semantic Intelligence).  
  
As long as my post are going I've just got a stack of (incoherent) "analysis" documents as the result of my work. And I had only those until now because I was stuck because of the previously mentioned mistake (ah, and because of my Bipolar Disease maniac episodes...).  
  
I should try to describe applications instead and see how and where fuel should burn properly inside a motion vehicle to generate traction. Every semicolon I write is updated into my GitHub repository, so, sorry if you browse that "scrapbook" and you don't find anything even intelligible.  
  
First, I'll try to describe a "problem" (problem "spaces" in this case) and how a Purpose driven user Community achieves its Goal(s) by means of Goods, Products and Needs satisfaction (ontology levels: from abstract upper ontology to user gesture command in user interface / service invocation).  
  
The problem is to organize interdisciplinary (multiple domains) Task(s) in a Purpose fulfilment network with Actors, Contexts and Roles (with attributes and values). Problem spaces (domains) are declaratively stated by DCI[1] design pattern: Data / Context / Interaction use cases definitions and instances.  
  
Collaborative Federated Actor network complying determinate Profile(s) satisfying specific Product / Good / Need abstraction playing determinate Role in use cases Context.  
  
Domain Translation between business domains, example: orders, delivery, invoicing (micro) services Model instances are the means by which distributed disparate data, schema and behavior of different sources (applications, services) integration could be performed by means of Semantic Intelligence and Augmentation Protocol(s).  
  
A domain can be defined in terms of a set of actions / tasks with the Purpose of satisfying some Goal solving the Need for a Good producing / gathering a Product. Ontology. Purpose as Goal “class”.

The principal focus is to deploy a (social) Collaborative peer (Actor) network for which entities and individuals develop Profile(s) which acquaint them with Purpose resolution capabilities. Then, according peer’s specific needs (domain Goals) the application orchestrates interactions needed for Product(s) Task(s) accomplishment.

Ontology:

Domain / Actor / Context / Role / Product / Good / Need / Purpose / Task / Goal / Exchange.

Domains: data, schema and behavior of business applications (ERP, CRM, BI, SCM, HMS, etc.).

General purpose business domains problem resolution / tasks, goals accomplishment helper tools.

Syndication (contextual hypermedia activation): QA. Polls. Learning. Profiles. Guided task (wizards), guided editors: Context: Goal / Purpose.

Contents: Wiki view of augmented knowledge. Addressing. Hypermedia. API (Wiki) render nodes / links semantically browseable.

Backend: Nodes / Protocol.  
  
SoLiD:

<https://solid.mit.edu>

DIDs (Blockchain dApps):

<https://w3c-ccg.github.io/did-spec/>

<https://ont.io/#/>

Executable models (flows): testing results, prompts, scoring.

Applications (use / implement like):

Drive / Jira / Trello / Keep / Mural / Tasks / Calendar.  
  
Ontology levels abstractions (data, schema, behavior): service / user interface rendering (activation).   
  
Dashboards components (widgets / media / extended content types / addressing).  
  
Hypermedia Activation. Addressing. Link extended content types resources elements / parts with other resources addressed elements.

Objectives:

Develop Protocol (APIs) to facilitate Enterprise Application Integration (EAI) by means of Semantic technologies and Machine Learning. Ontology matching driven data, schema, behavior inference / aggregation / matching. Reasoning and learning over different consolidated backends alignments.  
  
Distributed P2P (Blockchain) approach of data synchronization between peers for ease of deployment patterns election and datasources integration (APIs, microservices, etc.).  
  
Data alignment:  
  
Determine if two instances (example: records) of two different backends or services refer to the same entity (Customers : John D. / Employees : John Doe).  
  
Schema alignment:  
  
Determine, for example, meaning and equivalences between diverse (aggregated / composite) schemas (equivalent classes, equivalent attributes, equivalent roles).  
  
Behavior alignment:  
  
Determine meaning and equivalences between (aggregated / composite) behavior contexts and behavior contexts invocations / interactions (Appointment / Interview, anAppointment / anInterview. Behavior flows aggregated from backends / services learning).

Alignments Augmentations:  
  
Activation: type inference : classification (determine class / metaclass / roles for entity attributes and values).  
  
Activation infer attributes / relations : clustering (from multiple occurrences of same entity in diverse data sources).  
  
Aggregation: infer roles in contexts: regression (Person class in Employment interaction : Developer role).  
  
Integration of addressable resources. Reactive I/O (sync back ends). Content type driven semantic augmentation / annotations.

Augmentation of distributed resources. Annotations (Semantic / ML). API for resource / schema / interactions exploration / protocol for message based API "dialogs" execution. HAL (Hypertext Application Language), OData (REST) like interfaces.  
  
Example: Google Drive / Google Knowledge Graph APIs Augmented with ML / Semantic intelligence tailored for specific domains / application kinds.

Augmentation. Ontology matching. Hypermedia augmentation protocol. Browser / Client APIs.

URIs API for annotating network retrieveable resources metadata. Content type / model driven augmentations / activations (models features / outputs). Subject attributes / values. Occurrences contexts / roles. Paths, pointers, locators. Example: annotate document URIs (parts, sections, mentions), annotate images URI (whole image description, coords: classes, individuals), annotate DB, table, row, column, value URIs, annotate / describe service / APIs URIs. Hypermedia protocol composable with other (described / annotated) APIs / resources. Example: Drive APIs.

What my attempts are about where, in the beginning, to match different URIs or identifiers which refer to the same entity (in different databases / ontologies, for example) to perform some kind of "ontology matching".  
  
Then I've tried to develop a mechanism for using RDF Quads for encoding an object graph (and a layers class hierarchy) using Contexts to denote the class of an instance, Subjects to denote class instances and attributes (members) and values: Predicates / Objects.  
  
Quads are "reified" as Resource(s). Also, Resource is a functional wrapper reactive and event driven of an URI. And an URI could be implemented with whatever backend which could produce or consume events (databases, services, etc.). Resource layers hierarchy (Context) is to be implemented by an actor / role type object pattern.  
  
Then I've realized that some basic type inference could be performed with, for example, aggregating Subjects with the same predicates (Subject Kinds). Idem for Predicates, Objects and Contexts. I've also realized that plain "facts" statements could be aggregated in the previously mentioned class hierarchy to abstract further, from plain data, instance / class layers of what I call data / schema / behavior layers. Higher layers (i.e.: Behavior) "aggregate" lower layers.  
  
Layers shape is as follow:  
Resource : Functional URI wrapper.  
(Context : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);  
  
Each layer abstract:  
  
Statement (data instance):   
(Statement, Occurrence, Attribute, Value);  
someOne buys someProduct  
  
Entity (data class):  
(Entity, Statement, Occurrence, Attribute);  
someBuyer, someProduct (Entity);  
  
Role (schema instance):  
(Role, Entity, Statement, Occurrence);  
Buyer, Product (Role);  
  
Class (schema class):  
(Class, Role, Entity, Statement);  
Person, Good (Class);  
  
Flow (behavior instance):  
(Flow, Class, Role, Entity);  
someBought (Flow);  
  
Behavior (behavior class):  
(Behavior, Flow, Class, Role);  
Buy (Behavior);  
  
This "aggregations" are part of what I call "Augmentation(s)": Aggregation, Alignment and Activation are ones of those, which are functional transforms described declaratively in an object graph metamodel. The act of applying an Augmentation implies one source Resource (context), one template Resource (transform) and a resulting (set of) Resource(s).  
  
One also could Augment Resource(s) in a functional manner, using reactive event driven APIs so, for example applying "Person" class to "Employee" role could shield a Resource set of people being working for someone. The ultimate goal is to be able to "plug" as much "backends" connectors as posible into distributed peers which exposes protocols / APIs for knowledge driven hypermedia applications.

Implementation. Async / Reactive Service URIs / Connectors (sample):

Spark,

Lucene / Solr,

Kafka,

MQ,

ServiceMix,

Vert.x,

OSGi,

Spring Boot,

SCDF,

Jena, (RDFS, OWL, Turtle, N3, SPARQL),

Reasoning / Shapes,

JAF / JCA / JDBC / JNDI / JMX,

Metamodel / Teiid / D2RQ / OData,

HAL,

DIDs,

OpenShift (containers / deployment).

Application:

features / techniques / patterns.

Implementation deployment use cases. Sample Apps: SoLiD / PIM / PASCEN: App declaratively built with framework, Implementation Integrations.

[1] <https://en.wikipedia.org/wiki/Data,_context_and_interaction>

Workflows (Domain Goals) general purpose ontology matching integration framework.

The idea is basically that if you have: A) an ERP, B) a CRM, C) Drive or similar and D) an issue tracker / workflow / BPM or similar you see a "syndicated" dashboard that allows you to browse "annotated" hypermedia aligned and matched about the state of the processes in each of them.

After that if you act in A certain behavior the relevant / entailed changes are reflected in B, C and D (according to the processes and flows of the business domains of the applications that you have integrated) and that, through a “virtualized” model, have a homogeneous / aligned view (google JBoss Teiid, as an eventual back end, for example) of the domains you have.

All this through a back end of graphs (triple store), functional programming, "ontology matching", inferences and even Machine Learning in layers from the model to the functional view and various types of clients / APIs of applications or services.

Then there was the issue to export to a "standard" format (WS- \* / StratML, Swagger, GraphQL or an "Object Graph Mapper") the metadata that allows to consume or create endpoints / services / clients "declaratively" on the "inference" of what would be the APIs or interfaces and their flows (google REST HATEOAS or REST HAL) from what was integrated. All this goal / purpose driven: metadata about the processes / domains and their objectives.

There are particular cases (WS- \* / StratML, for example) that would serve both as "input" for the description of domains orchestration and for formatting the "output" of some type of description / interfaces.

### Features:

Data, Information, Knowledge exchange: data / schema / behavior Augmentation of virtualized and syndicated / aligned business domains. Business domains applications purpose / problem "spaces" interactions / translations. Addressable interactions: event sourcing. Purpose modelling: Business Domains.

Products And Services Community Exchange Network. Resource Oriented Knowledge Computing. Purpose driven Needs / Goods / Products Goals interactions.

Semantic Hypermedia Browser: declarative front-end / services. Forms / Flows. Annotation, Augmentation & other Domains.

Domains Workflow. Layer Contexts Domains. Domains Workflows (Domain Goals) general purpose ontology matching integration framework.

Levels: Upper / Onto Matching: reify Resource upper layers as Resource and aggregate into lower layers. Reified Entity, Relationship, Flow, Domain as upper layers and aggregated downwards (Rules / Productions). Productions dataflow (domain / range). From UI Gesture to backend operations.

### Domain Component Model:

Runtime configured (model: triple store, controller: object layer, view: functional layer) for Message parsing and Augmentations executions.

#### Model:

Triple Store. Meta Model Schema (RDF / RDFS). Upper Ontology. Primitives.

**Meta Model:**

Layers class hierarchy. Relationship : Relation / Relation kindOf Relationship.

Layer instances, classes, metaclasses, contexts, occurrences shape mappings (layers transforms / message augmentation templates) grammar shapes.

Reified Statement (Relation) Mappings: Model shapes / grammar metadata. Augmentations reify to / from meta model mappings / ontology statements.

Instance : URI

Class : Instance

Context : Class

(Context, Class, Instance);

Metaclass (Kind / Role) : Context

(Metaclass, Context, Class, Instance);

Occurrence : Metaclass

(Occurrence, Metaclass, Context, Class);

Statement (Relation) : Occurrence

(Statement, Occurrence, Metaclass, Context);

Relationship : Relation

Flow : Relationship

Domain : Flow

Model : Domain

##### Meta Model Quads schema:

URI (Predicate).

Statement : URI (Mapping).

(URI, URI, URI, URI);

Value : Statement (Function).

(Value, Occurrence : URI, Attribute : URI, Value : URI);

Sign : Value

(Sign, Value, Occurrence : URI, Attribute : URI);

Object : Sign

(Object, Sign, Value, Occurrence : URI);

Context : Object

(Context, Object, Sign, Value);

Kind : Context

(Kind, Context, Object, Sign);

Resource : Kind

(Resource, Kind, Context, Object);

Relation : Resource

(Relation, Resource, Kind, Context);

Relationship : Relation

(Relationship, Relation, Resource, Kind)

Flow : Relationship

(Flow, Relationship, Relation, Resource);

Domain : Flow

(Domain, Flow, Relationship, Relation);

Model : Domain

(Model, Domain, Flow, Relationship);

Data Layers:

Context Layer (reified) aggregation / population to / from aggregated / populated Object, Sign, Value layers (Data layers) Ontology (upper) matching: Predicates / Mappings / Functions. Contexts matching Occurrences / Attributes / Values.

Schema / Information Layers:

Kind, Resource, Relation Aggregated / Aligned Layer contexts instances.

Behavior / Knowledge Layers:

Relationship, Flow, Domain Aggregated / Aligned Layer contexts instances.

Domains Context Services:

REST / HATEOAS / HAL Context / URI backend API interface. Dataflow Context streams (domain / range). Layer Context type / instance type (service selector) events (Message) driven interface Augmentation (service features) domain streams signature request / response. Integrated Models (domain / services) participate in dataflows. Message Resolution: Model (integrated / contexts), Message prompts, dataflow domain context service / URI. Sync Model Contexts (i.e.: response matching domain / range of persistence contexts / services.

* Hierarchy: upper layers reify / render lower layers. Augmentations populate / aggregate lower / upper layers. Example: Domain Service populates data layers from context layers aggregations / aggregates Context instances from data layers.
* Message matching flows from more specific Layer instances (Statement predicates: Model onwards, URIs if none matchs / general upper ontology reified layers defaults) until final Context instance found with matching Message Predicates (selector: layer / type / instance). Matching Context Layer instance SPOs: aggregate / augment more general Layers Contexts instances. Matching Context Layer instance CSPs: aggregate / augment more specific Layers Contexts instances. Layers Context types aggregation signatures.
* Domain services dataflow: Contexts instances invoked according domain / range signatures. Events: LayerType::onMessage (matches / next layer). Traversal aggregates matchings from previous layers until aggregated Contexts.
* Context: Layer type / instance matching selector Context. Domain Services invoked passing Message as argument:
* Object : yields Aggregation statements. Contexts / Occurrences domain / range.
* Sign : yields Activation statements. Occurrences / Kinds domain / range.
* Value : yields Alignment statements. Kinds / Resources domain / range.
* Domain Service behavior: Context instance URI APIs. IO: Message. Domain features / busines logic (express declaratively: TBD). Process invocation result Message(s) recursively (dataflow). Hierarchy Context matching: reified URIs / Contexts.

#### Controller:

Resource Layers object hierarchy API.

Controller: Resource Layers object hierarchy API. Named Transforms (Resource URI Service interface / implementation bindings) dataflow: signatures pipelines. Triple Store object graph (DTOs).

Functional Layers Domain model / transforms (events / controller). Named Transforms (Resource URI Service interface / implementation bindings) dataflow: signatures pipelines.

##### Triple Store object graph (DTOs / OGM):

URI

Statement

Value

Sign

Object

Context

Kind

Resource

Relation

Relationship

Flow

Domain

Model

Controller: DTOs:

Aggregated super Contexts occurrences collection.

Statements expanded view (occurrences traversal).

Transforms:

DTO: Mapping (Function). From DTO Layer type to DTO Context Layer occurrences type. Stateful Transforms.

TBD.

#### View:

Layers Domain hierarchy (Functional API).

Monads AST / Parser Builder. Monads: Parsing / Matching, Zippers. Introduction. API: Augmentations, Transforms / Mappings. Traversal. Dataflow.

Layers Monads / Parser Monads (Messages : Rules / Productions). Functional events dataflow (selector signatures : Layer instance Activation).

Domain Declarations: populate layers from Semiotic Context layer Domain description layers resources: Service Resources I/O layers matching / producing semiotic statements for Domain I/O.

##### DOM / AST Hierarchy:

URI<URIDTO>;

Statement<StatementDTO>;

Value<ValueDTO>;

Sign<SignDTO>;

Object<ObjectDTO>;

Context<ContextDTO>;

Kind<KindDTO>;

Resource<ResourceDTO>;

Relation<RelationDTO>;

Relationship<RelationshipDTO>;

Flow<FlowDTO>;

Domain<DomainDTO>;

Model<ModelDTO>;

Functional Model Monads wraps Meta Model Layers DTOs which represents an endpoint / interface for its Resource URI type and instance.

View. Monads:

Layer Monad::of(Layer Monad hierarchcally compatible DTO);

Layer Monad::flatMap(Layer hierarchically compatible DTO) : DTO Context Layer occurrences type Layer Monad.

Layer Monad::value : DTO;

### Augmentations:

Domain Controllers (DTO) handled Functional Model API Selector / Transforms.

Matching: Selectors.

Traversal: Layers Contexts.

Matching / Traversal: Resource DTO implements Role / Predicate (Role / Predicate Function / Mapping). Aggregated Resources set specification : Message (model / input prompts).

Transforms: Resource DTO implements Mapping / Function.

Augmentations:

Aggregation: Occurrences (clustering). Flat Map of current DTO type.

Activation: Attributes (classification). Flat Map of lower layer DTO type.

Alignment: Values (regression). Flat Map of upper layer DTO type.

TBD.

* Model: higher Model / Functional layer.
* Augmentation Domain:
* Message:
* (Selector : Statement, Aggregation : Statement, Activation : Statement, Alignment : Statement);
* Model (DTOs). Addressing (visitor):
* Model::onMessage(msg : Message) : Monads (Alignment stream)
* Augmentations (Monads);
* Monad::flatMap(dto : DTO) : Monad : functional / model hierarchies / aggregation;
* Browse / Traversal : Messages.
* Augmentation Domain Statements parses / emits Model / Message Statements.
* Aggregation Domain:
* (Domain, Layer, Rule, Occurrences layers);
* Activation Domain:
* (Domain, Occurrence, Rule, Attributes layers);
* Alignment Domain:
* (Domain, Attribute, Rule / Occurrence, Value / populated Layer statement);
* Dataflow.
* DTOs: Model / OGM. Mappings. Functions. Predicates.
* Monads: Functional View.
* Message: Controller / Command Transform API (Transforms Selector / Augmentations Roles / Predicates). Roles / Predicates encoding / addressing resolution.
* Selector: Monads from Selector Transform Role (stream). Role predicates matching / addressing Resources (Model visitor). Aggregated Message Resource set Monads (i.e.: Relationships / Relations). Layers Context instances selector.
* Selector: Select Context Layer instance.
* Aggregation: Rules / Productions. Monads from Aggregation Transform Role (stream): Flat map on Message Aggregation Role / Predicate over Selector stream. (i.e.: Flow Contexts having Relationship Occurrences having Relation Occurrences).
* Aggregation: Context instance types Aggregation criteria (Role / Predicates; filter / join / prompt: Model / Message deferred resolution). Aggregation: (Context, Occurrence, Attribute, Value) mappings / axes aggregation predicates.
* Aggregation: Example: Criteria for which a Statement instance is occurrence in a (new aggregated) Relation instance and a Relation instance is occurrence in a (new aggregated) Relationship instance. OGM (DTOs) and Context instance types (RDFS). Contexts types schema and instance types schema / shapes.
* Aggregation: Assert Context Layer instance occurrences in next Context Layer (recursively). Inheritance levels polymorphic Aggregation (i.e.: aggregate Relationship as a Relation).
* Aggregation: Shapes / Parsing / Traversal: Monad Zippers / Predicates.
* Activation: Layers Contexts types instance types. Context occurrences (aggregation) Kinds.
* Activation: Types / Attributes Monads from Activation Transform Role (stream): Flat map on Message Activation Role / Predicate over Aggregation stream. Aggregated statements instance type Attributes. Statement predicates: previous context statements subjects (filter / join / prompt: Model / Message deferred resolution).
* Alignment: Attributes / Values. Monads from Alignment Transform Role (stream): Flat map on Message Alignment Role / Predicate over Activation stream. Activated statements Attributes Values. Statement objects: previous context statements predicates (filter / join / prompt: Model / Message deferred resolution).
* Dataflow: Domains Layers Mappings. Further Augmentations (Messages). Activation values matches Domain subject signatures. Domain activation Layer emits new Resource set Messages. Augmentations according Model / Domain.
* Augmentation Domain Statements. Model Messages: Core Augmentations. Invoked after each Model Message processing. Layers. Initially matches Upper Ontology / Core Domains.

#### Aggregation:

Clustering stream. Registry. Attribute Roles in Contexts. Populate schema quad layers.

Productions: Layers down through the contexts hierarchy are "productions" of previous layers.

Rules: Layer contexts aggregates previous layers contexts as their subjects matching / aggregating same subjects / predicates / objects.

Layers conform a hierachy of which Value is root and Domain is the last Layer in the inheritance chain.

Layers Resource Context / DOM API. Levels (inheritance hierarchy reification). Upper / Lower Layer Roles. Transforms. Bindings (contexts resolution by reference model matchings).

#### Activation:

Classification stream. Naming. Attribute Types in Contexts. Populate model Kinds.

Kinds: (Context : Kind, Resource, Attribute, Value);

Hierarchies: Resource reified Kind as Kind Resource (sub Kind).

Kinds layouts:

(S, P): OK; (P, O): SK, (O, S): PK; (SK, OK): CK;

Role: Sets. Layers CSPO Resource types.

Type Promotion (roles). Order (dataflow). Reified Relation / Relationship (Production / Rule) context roles / interactions. Matching.

#### Alignment:

Regression stream. Index. Attribute Values in Contexts. Align / complete missing information.

Model reification: Role Context. Addressable Augmentations (Object extension which is result of Context intension).

Kinds: Streams of corresponding Roles.

Semiotic Layer: (Augmentation, Subject, Predicate, Object);

Reference Model: Map Reduce. Reified Layers. Levels.

### Message I/O:

Reactive event driven Message Roles / Predicates matching / processing / emission of aggregated matching Resource set results.

Message: Role / Predicate Context matching. Resource set specification.

Parse Message as corresponding Layer Context DTO / Functional wrapper and filter interaction Resource set according Role / Predicate matching. Message resolves to inputs (model prompts) and parameters (client prompts) to be populated and augmented from initial matching layer instance and is populated back with an aggregated response of which augmentations gave as result.

Message Role / Predicate matching Resource set: Perform Augmentations as with source Domains data (inputs / parameters) of new (prompts) / existing Contexts / Occurrences (Aggregation), of new (prompts) / existing Attributes (Activation) and of new (prompts) / existing Values (Alignment) until there are no further Augmentations (dialogs).

Selectors: DTOs implementing Predicate interface.

Transforms: DTOs implementing Function interface.

Streams:

Kinds. Filter: Predicate.

Contexts, Occurrences, Attributes, Values: Productions of Predicates according CSPO role application.

Dataflow: Result Transform matching rules signatures of Semiotic graph domain / range.

Build Message graph via navigation of the model (Forms / Flows HATEOAS APIs, domain / range dataflows). Transform mapping: Message prompts: resolve from model / prompt client.

TBD.

* Model: higher Model / Functional layer.
* Augmentation Domain:
* Message:
* (Selector : Statement, Aggregation : Statement, Activation : Statement, Alignment : Statement);
* Model (DTOs). Addressing (visitor):
* Model::onMessage(msg : Message) : Monads (Alignment stream)
* Augmentations (Monads);
* Monad::flatMap(dto : DTO) : Monad : functional / model hierarchies / aggregation;
* Browse / Traversal : Messages.
* Augmentation Domain Statements parses / emits Model / Message Statements.
* Aggregation Domain:
* (Domain, Layer, Rule, Occurrences layers);
* Activation Domain:
* (Domain, Occurrence, Rule, Attributes layers);
* Alignment Domain:
* (Domain, Attribute, Rule / Occurrence, Value / populated Layer statement);
* Dataflow.
* DTOs: Model / OGM. Mappings. Functions. Predicates.
* Monads: Functional View.
* Message: Controller / Command Transform API (Transforms Selector / Augmentations Roles / Predicates). Roles / Predicates encoding / addressing resolution.
* Selector: Monads from Selector Transform Role (stream). Role predicates matching / addressing Resources (Model visitor). Aggregated Message Resource set Monads (i.e.: Relationships / Relations). Layers Context instances selector.
* Selector: Select Context Layer instance.
* Aggregation: Rules / Productions. Monads from Aggregation Transform Role (stream): Flat map on Message Aggregation Role / Predicate over Selector stream. (i.e.: Flow Contexts having Relationship Occurrences having Relation Occurrences).
* Aggregation: Context instance types Aggregation criteria (Role / Predicates; filter / join / prompt: Model / Message deferred resolution). Aggregation: (Context, Occurrence, Attribute, Value) mappings / axes aggregation predicates.
* Aggregation: Example: Criteria for which a Statement instance is occurrence in a (new aggregated) Relation instance and a Relation instance is occurrence in a (new aggregated) Relationship instance. OGM (DTOs) and Context instance types (RDFS). Contexts types schema and instance types schema / shapes.
* Aggregation: Assert Context Layer instance occurrences in next Context Layer (recursively). Inheritance levels polymorphic Aggregation (i.e.: aggregate Relationship as a Relation).
* Aggregation: Shapes / Parsing / Traversal: Monad Zippers / Predicates.
* Activation: Layers Contexts types instance types. Context occurrences (aggregation) Kinds.
* Activation: Types / Attributes Monads from Activation Transform Role (stream): Flat map on Message Activation Role / Predicate over Aggregation stream. Aggregated statements instance type Attributes. Statement predicates: previous context statements subjects (filter / join / prompt: Model / Message deferred resolution).
* Alignment: Attributes / Values. Monads from Alignment Transform Role (stream): Flat map on Message Alignment Role / Predicate over Activation stream. Activated statements Attributes Values. Statement objects: previous context statements predicates (filter / join / prompt: Model / Message deferred resolution).
* Dataflow: Domains Layers Mappings. Further Augmentations (Messages). Activation values matches Domain subject signatures. Domain activation Layer emits new Resource set Messages. Augmentations according Model / Domain.
* Augmentation Domain Statements. Model Messages: Core Augmentations. Invoked after each Model Message processing. Layers. Initially matches Upper Ontology / Core Domains.

### Component Domains:

Functional event driven Domains configures models behaviors. Domain types: service resources, interfaces (transforms), signatures (dataflow).

I/O / Persistence Domain.

Sets Augmentations Domain.

FCA Augmentations Domain.

Endpoints I/O Domain.

Predictions Domain.

Dimensional Domain.

Registry Domain.

Index Domain.

Naming Domain.

Business Domains: business specific domain types.

#### I/O / Persistence Domain:

Events (event sourcing). Backends. Peers. DIDs.

Semiotic (Functional Message Signature): (PersistenceType, PersistenceSubject, PersistenceMember, PersistenceValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain persistence resource types (employee).

Member Kind: persistence members resource types (employee/salary;ARS).

Value Kind: range resource types (salary;ARS).

Reify Persistence semiotic predicates as Relationship Relations (Values as Relation Resources). Align domain / range with domains / primitive types (Member Kind, salary;ARS).

Event sourcing:

(PersistenceContext, PersistenceContext, PersistenceMember::new, PersistenceSubject);

(PersistenceContext, PersistenceContext, PersistenceMember::delete, PersistenceSubject);

(PersistenceContext, PersistenceSubject, PersistenceMember::delete, PersistenceSubject);

#### Sets Augmentation Domain:

Semiotic mappings population. Augmentations: Aggregation (layers), Alignment (ontology), Activation (layers dataflows). Render Resource hierarchies.

Semiotic (Functional Message Signature): (SetContext, SetParent, SetSubject, SetChildren);

Context Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Parent Kind: domain resource types (resource).

Subject Kind: attribute resource types (resource/resource).

Children Kind: range resource types (resource).

Reify Sets semiotic predicates as Relationship Relations (Values as Relation Resources).

Sets API: Augmentations, Transforms / Mappings. Traversal (Context, Statement, Kind, Resource).

#### FCA Augmentations Domain:

Semiotic mappings population. Augmentations: Aggregation (layers), Alignment (ontology), Activation (layers dataflows). Objects / attributes objects / concepts traversal. Render Resource hierarchies.

Semiotic (Functional Message Signature): (FCAContext, FCASubject, FCAAttribute, FCAValue);

Context Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain resource types (resource).

Attribute Kind: attribute resource types (resource).

Value Kind: range resource types (resource).

Reify FCA Context semiotic predicates as Relationship Relations (Values as Relation Resources).

FCA API: Augmentations, Transforms / Mappings. Traversal (Concepts, Objects, Attributes, Products).

FCA / VSM (Vector Space Model) Meta Model Context Encoding:

Attributes: Resource URIs. Polygon side lengths (class).

CSPO Roles (scaling): polygon sides (metaclass).

CSPO scaling: ordered side position.

Polygon sides dot-notation ordered sides lengths: Resource Layer Statement IDs (instance).

Sides dot-notation sum: side in context (occurrence).

Normalization: Resource URI attributes embeddings / primes quad polygon sides lenghts.

Nested Resource encoded attribute values (layers hierarchy): sides lengths concatenation (ordered dot notation) sum (occurrence).

Graph navigation (layers / transforms: concepts / objects containing / contained in concepts / objects attributes IDs / lengths).

FCA Contexts. Encoding. Flows. Order (types: dataflow signatures domain / range, instances: dimensional attributes).

#### Endpoints Domain:

Streaming I/O Dataflow.

Semiotic (Functional Message Signature): (EndpointContext, EndpointSubject, EndpointRequest, EndpointResponse);

Context Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain resource types / referrer (employment).

Request Kind: request resource types (person).

Response Kind: response range resource types (employee).

Reify Endpoint semiotic predicates as Relationship Relations (Values as Relation Resources).

Augmented Resources Contexts / Interactions Services.

Forms / Flows (Grammar / Protocol Builder. Prompts). Resource augmentation endpoints. Forms / Flows browsing APIs. DCI: Declarative Forms / Flows.

OGM / Client Drivers Services.

REST: Current / referrer. Rel. HREF. Link body. Metadata. Endpoint Domain.

Monads: Reify available Transforms as activable Resources (Function addresses). REST / HATEOAS HAL.

Prompts / Dialogs: Function arguments (values / options) shown as link addresses in Transforms navigation Flows. Activation browse of Resources in Transform context.

Chained Activations for complete contexts resolution / flows. Complete layers productions rendering / navigation from higher to lower layers.

Browse: request address content representation (extracted from current state) embedding current state representation as request context body. Model matches address and returns augmentation using request body as argument / context.

#### Predictions Domain:

Semiotic (Functional Message Signature): (PredictionType, PredictionSubject, PredictionItem, PredictionValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain resource types (image).

Item Kind: prediction resource types (image/face).

Value Kind: range resource types (face).

Reify Prediction semiotic predicates as Relationship Relations (Values as Relation Resources).

#### Dimensional Domain:

Semiotic Layer: (DimensionType, DimensionSubject, DimensionItem, DimensionValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O. Time example (contains / before).

Subject Kind: domain resource types (hour; dayOfWeek) : 1. Monday.

Item Kind: dimensional resource types (hour/minutes; dayOfWeek/dayOfWeek) relations: contains / before.

Value Kind: range resource types (minutes; dayOfWeek) : 60. Tuesday.

Data / Key Value: Price.

Information / Schema: Tuples. Price variation.

Knowledge / Behavior: Values relations. Monthly price increase.

Upper Ontology: relations / primitives.

Reify Dimension predicates as Relationships Relations (Values as Relation Resources).

Example: Marriage.

Predicates:

:aHusband :marriedWith :aWife

:marriedWith rdfs:domain :Male

:marriedWith rdfs:range :Female

Relationship:

(aMarriage : Relation, anStatement : marriageStatement, aKind : husbandRole, aResource : aHusband);

(aMarriage : Relation, anStatement : marriageStatement, aKind : wifeRole, aResource : aWife);

(Marriage : Relationship, Marriages : Relation, anStatement : marriagesStatements, aKind : marriageRole);

Predicates / Relationships, Relationships / Predicates entailment. Dimensional: inference / relation types / restrictions.

Encode order / hierarchies / relations (parent / child, prev / next, etc.) / iterations / conditionals / jumps.

Dimensional Domain: dimensions, units, measures, values. Comparisons, relations. State. Events (marriage example). Verbs (action, passion, state). Order (data / schema / behavior).

#### Registry Domain:

Key / Value for graph contexts, nodes, predicates.

Semiotic (Functional Message Signature): (RegistryType, RegistrySubject, RegistryKey, RegistryValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain node resource types (person).

Key Kind: registry resource types (person/age;int).

Value Kind: range value resource types (age).

Reify Registry semiotic predicates as Relationships Relations (Values as Relation Resources). Align domain / range with primitive types (Key Kind, age;int).

#### Index Domain:

Indexing of graph contexts, nodes, predicates.

Semiotic (Functional Message Signature): (IndexType, IndexTerm, IndexScope, IndexValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Term Kind: domain node resource types (resource).

Scope Kind: dimensional resource types (resource/resource).

Value Kind: range value resource types (resource).

Reify Index semiotic predicates as Relationships Relations (Values as Relation Resources).

#### Naming Domain:

Terms translation in contexts for graph contexts, nodes, predicates. Alignment / matching.

Semiotic (Functional Message Signature): (NamingType, NamingSubject, NamingContext, NamingValue);

Type Kind: Domain Service Handler. Domain signatures (domain / range: Subject Kind / Object Kind). Domain graph mappings context handler: function P(S) : O.

Subject Kind: domain node resource types (term).

Context Kind: dimensional resource types (term/term).

Value Kind: range value resource types (term).

Reify Naming semiotic predicates as Relationships Relations (Values as Relation Resources).

### Domains Dataflow:

Layers Dataflow: Augmentation. Rules / Productions matching (Reference Model / Kinds Aggregation).

Semiotic Dataflow: Object Kind matches Subject Kind of Context Kind signatures. (Sucessive Layers Dataflow).

### Ontology Matching:

Upper Ontology. Grammars. Primitives.

Matching: Resource occurs as context / occurrence / atribute / value or class / occurrence / context / metaclass / instance in equivalent occurrence contexts (kinds / order / shapes / type hierarchies).

Meta Model encodes mappings for equivalence / relations hierarchies for entities instance occurrences in roles in contexts for concepts recursively till upper onto / primitive terms / relations.

Reify relation from / to predicates (semiotic) / relation entity (expanded relation entity / roles statements). Dimensional measures / state events. Shapes: transforms / rules.

FCA Ontology Matching: Upper ontology / primitives. Reference Model objects / attributes encoding. Encoding (scaling): lattice concepts relations / transforms traversal.

Semiotic mappings population. Augmentations: Aggregation (layers), Alignment (ontology), Activation (layers dataflow transforms: context products).

Reference Model Contexts.

Meta Model Layers Contexts.

Alignments (Reference Model types / values):

Data Alignment: key / val.

Schema / Information Alignment: tuples.

Behavior / Knowledge Alignment: dimensional.

### Appendix:

#### Reference Model:

##### Encodings:

(Type, Object) Key / Value.

(Context, Subject);

(Subject, Predicate);

(Predicate, Object);

Reference Model: Key / Value de-referenceable (for matching / embedding purposes) URIs having as host the peer that identified the Resource. DIDs resolution. Cons cells encoding for Domain Component Model I/O.

Quads Encoding: (Context, Occurrence, Attribute, Value). Kinds. OGM. Sets / FCA. Context, Metaclass / Role, Class, Instance, Occurrence (Context).

Matching: Addressable / Browseable Encodings (FCA / Sets) / Identifiers. Order, Dimensional / hierarchical relations (attribute sets). Root Layers (Reference Model) traversal.

Functional: Selectors (TBD)

#### FCA Domain Contexts:

Semiotic (Functional Message Signature): (FCAContext, FCASubject, FCAAttribute, FCAValue);

##### Reference Model Context:

FCA Lattice (concepts / objects / attributes): (types / values) x (types / values). Encoding (IDs): ontology matching enabling type / instance calculations / traversal / transforms.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Value | Value | Value | Value |
| Object | X |  |  |  |
| Object |  | X |  |  |
| Object |  |  | X |  |
| Object |  |  |  | X |

##### Meta Model Context:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Resource A | | | |
|  | Context | Subject | Predicate | Object |
| Resource B | X |  |  |  |
| Resource C |  | X |  |  |
| Resource D |  |  | X |  |
| Resource E |  |  |  | X |

Encoding: FCA Scaling. FCA Context objects and attributes are corresponding CSPO Contexts types scaling enclosed Context types instances. A potential encoding of axes objects and attributes (rows and columns) would be a bitstring of length 4 x n, being n the length of an instance identifier for each quad Context encoded in its corresponding bitstring quad space (4 is for CSPO quad types instances identifiers segments). Then, navigation should be allowed from a pair of object / attribute to another object / attribute: (type, object) x (type, object): (type, object).

##### Layers Context:

For each layer context statement build tables which axes correspond to each context CSPO context types. Aggregate CSPO types / values in the form shown below (nested contexts).

Values intersections are instances of corresponding types. Example:

(RoleA x StatementB: KindC) : Kind RoleA plays in StatementB.

Rules are of the form:

(TypeA, TypeB) > AggregatedKindResources;

For example, in Relation lattice:

(someRoleA, someKindB) > AggregatedRelationResources;

Relation matrix:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Relation | Kind | Role | Statement |
| Relation | Relation | Kind | Role | Statement |
| Kind | Kind | Relation | Statement | Role |
| Role | Role | Statement | Relation | Kind |
| Statement | Statement | Role | Kind | Relation |

Matrices for other layers (Kind, Role, Statement, Resource, Context) contexts follows the same principles.

Nested Contexts:

Reifying one aggregated layer SPO layer (for example: Kind in the previous table) has original context matrix axes in the corresponding SPO layer (Subject in this case):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Kind | Relation | Statement | Role |
| Kind | Kind | Relation | Statement | Role |
| Relation | Relation | Kind | Role | Statement |
| Statement | Statement | Role | Kind | Relation |
| Role | Role | Statement | Relation | Kind |

The purpose of this is to retrieve enough concepts (FCA) metadata to populate concepts / objects / attributes conforming a Lattice of related Resources and those relations values (as in the above example).

Layout: The aggregated statements have as Contexts the occurring SPOs in a Context layer statement and its SPOs are the occurrence Context and the other SPOs in the occurring statement. For a Context in an aggregated statement occurring as (SPO) in the occurrence statement, occurrence statement Context is its (SPO) and its aggregated (SPO) is occurrence statement (SPO).

Layout: Having a Context layer, a matrix (FCA context) of the form (CSPO x CSPO) is built for aggregation of models. The aggregated statements (rows / columns) have as Contexts the CSPOs (occurrences) of an axis and and its SPOs are given from the types / values of the context layout. For an aggregated Context statements / matrix, the original context is located in the (SPO) axis from the (SPO) which it was taken from the original Context matrix.

Augmentations: TBD.

Aggregation:

Activation:

Alignment:

Transforms (products / encoding) Dataflow:

Use cases:

Use FCA Lattice for sorting / ontology matching / augmentations / query / ontology browsing.

Aggregation: Complete contexts objects / concepts / attributes by FCA / inference.

Inference example: (Statement x Statement): Relations between both Statements.

Learning: ML embeddings for types / values / concepts.

TBD: (metaclass, class, occurrence, instance) relations / atttributes.

TBD: Set oriented intension (C) / extension (O) and relations between sets.

TBD: Discover IDs / encoding techniques enabling algorithmic translation of models operations.

#### Sets Domains:

Semiotic (Functional Message Signature): (SetContext, SetParent, SetSubject, SetChildren);

Augmentations:

Aggregation:

Kinds: (Context : Kind, Resource, Attribute, Value);

Hierarchies: Resource reified Kind as Kind Resource (sub Kind).

Kinds layouts:

(S, P): OK; (P, O): SK, (O, S): PK; (SK, OK): CK;

Role: Sets. Layers CSPO Resource types.

Layers Aggregation:

Productions: Layers down through the contexts hierarchy are "productions" of previous layers.

Rules: Layer contexts aggregates previous layers contexts as their subjects matching / aggregating same subjects / predicates / objects.

Activation:

Resource Context: Kind.

Alignment: Kind Attribute / Value Resource Statements. Shapes (inference of Attribute Value by context, class, metaclass, instance occurrences.

Transforms. Dataflow.

Sets Layout and encoding bitstring mask format:



#### To Do / Items:

* Test Browser (Forms / Flows HATEOAS debug console):
* Facets: Browse / Stream Model.
* Model: Browse / Stream.
* Messages: Build / Augmentation Response (Browse / Stream Model).
* Patterns:
* Master / Detail.
* Inheritance.
* Categories. Ordered Sets. Order relation. Inclusion properties / relations.
* Item: Inventory / Product. Order / LineItem.
* Others.
* Patterns: Alignment.
* Pizza ontology:
* For each ingredient calculate total price.
* For average sales calculate ingrediens orders amount, actual / projected revenues.
* Architecture:
* Context instance of Layer type and of Layer instance type (Relation / Marriage). Metaclass, class, instance, contexts / occurrences, roles, attributes, values members from Model.
* Domain I/O: URI REST / HATEOAS / HAL URI API interface (OGM / Domains / IO). TBD.
* DTOs OGM Interfaces:
* URI : Predicate
* Statement : URI. Mapping (Matching URIs). Occurrences.
* Value : Statement. Function (Contexts Subjects matching Predicates). Contexts.
* Resource: Value. Aligned URIs.
* Context: Aligned Statements. Domain: map selector, map aggregation, map activation, map alignment.
* StatementMonad::of(domainUri : URI);
* StatementMonad::flatMap(stmt : Statement) : StatementMonad<Statement> (model / message / backend I/O prompts).
* Message:
* Message (URIs quad) flows from upper layer (URI Statements which reifies all aggregated / sub context types) matching downwards until finding correct context instance.
* Statement Context domains perform relevant contexts streams / I/O (model / message / backends prompts).
* Message:
* (Context / Selector, Occurrence / Aggregation, Attribute / Activation, Value / Alignment);
* Domains (Contexts Mappings / Transforms):
* Domain / mapping function / inference / IO invocation. Dataflow (domain / range).
* LayerMonad.of(LayerContextDTO);
* Flat Map: DTO Mapping Transform.
* Model::flatMap(Selector)
* flatMap(Aggregation) : Aggregated Occurrences.
* flatMap(Activation): Occurrence Attributes.
* flatMap(Alignment) : Attribute Values.
* Data layer: Context Layer (reified) aggregation / population aggregates / populates Object, Sign, Value layers (Data layers) Ontology (upper) matching: Predicates / Mappings / Functions. Contexts matching Occurrences / Attributes / Values.
* Schema / Information Layers:
* Kind, Resource, Relation Aggregated / Aligned Layer contexts instances.
* Behavior / Knowledge Layers!
* Relationship, Flow, Domain Aggregated / Aligned Layer contexts instances.
* Domains Context Services:
* REST / HATEOAS / HAL Context / URI backend API interface. Dataflow Context streams (domain / range). Layer Context type / instance type (service selector) events (Message) driven interface Augmentation (service features) domain streams signature request / response. Integrated Models (domain / services) participate in dataflows. Message Resolution: Model (integrated / contexts), Message prompts, dataflow domain context service / URI. Sync Model Contexts (i.e.: response matching domain / range of persistence contexts / services.
* Hierarchy: upper layers reify / render lower layers. Augmentations populate / aggregate lower / upper layers. Example: Domain Service populates data layers from context layers aggregations / aggregates Context instances from data layers.
* Message matching flows from more specific Layer instances (Statement predicates: Model onwards, URIs if none matchs / general upper ontology reified layers defaults) until final Context instance found with matching Message Predicates (selector: layer / type / instance). Matching Context Layer instance SPOs: aggregate / augment more general Layers Contexts instances. Matching Context Layer instance CSPs: aggregate / augment more specific Layers Contexts instances. Layers Context types aggregation signatures.
* Domain services dataflow: Contexts instances invoked according domain / range signatures. Events: LayerType::onMessage (matches / next layer). Traversal aggregates matchings from previous layers until aggregated Contexts.
* Context: Layer type / instance matching selector Context. Domain Services invoked passing Message as argument:
* Object : yields Aggregation statements. Contexts / Occurrences domain / range.
* Sign : yields Activation statements. Occurrences / Kinds domain / range.
* Value : yields Alignment statements. Kinds / Resources domain / range.
* Domain Service behavior: Context instance URI APIs. IO: Message. Domain features / busines logic (express declaratively: TBD). Process invocation result Message(s) recursively (dataflow). Hierarchy Context matching: reified URIs / Contexts.