Distributed Integration and Consistency for Knowledge Semantics

Resource Oriented Knowledge Computing.

Ontology Matching (data, schema, behavior: reified operations as schema / data).

Model: Layers, Contexts, Aggregations.

Messages: IO Resources / Augmentations.

Event Sourcing. Dataflow.

Resource Oriented. Forms / Flows. Forms / Flows: HATEOAS HAL. MVC. REST. Meta DCI Models protocol based.

**Meta Model (DCI)**

DCI Meta Models:

Data (data): Relation hierarchy Model.

Context (schema / dataflow): Augmentation hierarchy Model. Model layers extends corresponding Data layers.

Interaction (behavior / services) Dialog hierarchy Model. Model layers extends corresponding Data layers.

Contexts and Interactions Models extending / reified as Data Model layers enabling matching, inference and augmentations (FCA / ML embeddings for example) for behavior and schema alignments.

**Meta Model (Data)**

**Reference Model:**

ID : URL;

Occurrence: Context;

Context : ID (Context / ID : intension, Object / Occurrence, Sign / Kind / Metaclass / Attribute, Value / Role / Class : extension);

Root of Meta Model hierarchy.

**Notation:**

[LayerType] : [LayerSuperType] ([ContextType], [SubjectType], [PredicateType], [ObjectType]);

**Layers:**

Resource : Context (Resource, Resource, Resource, Resource);

Statement : Resource (Statement: c, Resource, Resource, Resource);

Role / Class : Statement (Role: b, Statement, Resource / Attribute, Resource / Value);

Role / Class aggregating CSPO Resource (IDs) sharing Attributes for their Objects / Values.

Kind / Metaclass : Role (Kind: a, Role, Statement, Resource);

Kind: Aggregated similar Roles occurring as Resources (Object) in Statements (Predicate).

Relation / Entity : Kind (Relation, Kind, Role, Statement: c);

Rel type (Relationship) instance / bindings. An Entity (Relation: intension) and their Statements for its Kind / Role occurrences (occurrences: kinds / roles Relation plays in statements. Matching. Object: extension). Data (DCI)[1].

Mapping : Relation (Mapping, Relation, Kind, Role: b);

Rel players types / bindings scenarios. Information. Interaction (DCI)[1]. Mapping Role and Relation Kind: dataflow promoted types / order: relationships players domain / range. Entity alignment.

Relationship : Mapping (Relationship, Mapping, Relation, Kind: a);

Rel type declaration, player types. Knowledge. Context (DCI)[1].

Mapping and Relationship layer contexts are "calculated" (reifying) by Relation layer context kinds.

Semiotic Layer (ontology matching):

Value (Value, Value, Value, Value);

Sign : Value (Sign, Value, Value, Value);

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

Context : Object (Context, Object, Sign, Value);

Reference Model:

Root of MetaModel hierarchy.

**Dataflow Model (Context).**

Dataflow:

Iterations (types / kinds order declaration). Streams: Contexts / Occurrences Bus (signatures / discovery).

Conditionals (predicates / filters on types / kinds attributes / values) on Iterations.

Jumps (aggregation / stack sub-streams) on Conditionals. Apply Kinds on matching / referring Contexts (Employment, Person: Employee). Extract Kinds on matching / referring Contexts (Family, Father).

Order. Comparators: common upper hierarchies, Dataflow domain / range, SortedSet. Lattices (FCA contexts).

Augmentation: Aggregations, Alignments, Activations. Perform encoding dataflow.

Encoding: Augmentation, Template, Mapping, Transform.

OntResource: Ontology Matching (aligned URLs): semiotic context.

Dataflow Model:

Resource: (OntResource, OntResource, OntResource, OntResource);

Statement: (Transform, OntResource, OntResource, OntResource);

Role: (Mapping, Transform, OntResource, OntResource);

Kind: (Template, Mapping, Transform, OntResource);

Relation: (Augmentation, Template, Mapping, Transform);

Template: Reified Model and Model instances (hierarchies). Match inputs. Iterations.

Mapping: Dataflow reified operation flows bindings (subscriptions). Predicate / Object stream endpoints?. Conditionals.

Transform: Dataflow reified results. Jumps.

**IO / Services Meta Model (Interaction):**

Model for back ends synchronization and services exposures.

Context: (Model, OntResource, Resource, Resource); Key / value dictionary for source Model ontology matching.

Resource: (OntResource, OntResource, OntResource, OntResource); Full matched Resource descriptions: Type, ID, Attributes, Values.

Statement: (Assertion, OntResource, OntResource, OntResource); Transform (Jumps). Source Model(s) interaction interface Services URLs (IO).

Role: (Prompt, Assertion, OntResource, OntResource); Mapping (Conditionals). Predicates (LHS: Predicate, RHS: Object).

Kind: (Interaction, Prompt, Assertion, OntResource); Template (Iterations). Aggregate matching Interaction Assertion Prompts.

Relation: (Dialog, Interaction, Prompt, Assertion); Augmentation. State flows.

TOC.docx:

[Introduction 1](#_gjdgxs)

[Metamodel](#_30j0zll) 3

[Layers 5](#_1fob9te)

[Aggregation 8](#_3znysh7)

[Augmentation 9](#_2et92p0)

[Functors 9](#_tyjcwt)

[Messages 10](#_3dy6vkm)

[Dataflow / Activation 11](#_1t3h5sf)

[Services 12](#_4d34og8)

[Encoding 13](#_2s8eyo1)

[RDF Quads 15](#_17dp8vu)

[Semiotic Encoding 18](#_3rdcrjn)

[Normalization 20](#_26in1rg)

[Parsing 21](#_lnxbz9)

[Protocol 22](#_35nkun2)

[Addressing 23](#_1ksv4uv)

[Low level 23](#_44sinio)

[Hypermedia Activation 25](#_2jxsxqh)

[Facets / Levels 26](#_z337ya)

[Features / APIs 27](#_3j2qqm3)

[Features 29](#_1y810tw)

[Domains schema 29](#_4i7ojhp)

[Ontology matching 29](#_2xcytpi)

[Augmentations 32](#_1ci93xb)

[Deployment Use Case: Goals Application](#_3whwml4)

# Mision / Vision

## Objectives

## Description

# Use Cases

## Problem

Hypermedia Use Cases (Ontology Levels). Integration / Augmentation / Alignment / Annotation of distributed resources. (Augmented) Content type driven. Encoding / Addressing (links / browse / parts / rels / roles). Microformats (embedding). Wiki like abstract representation (DAV / Forms / Flows / indexes).

## Solution

# Approach

Integration by Augmention. Integration by Extension Declarative Application Design.

Features / Approach: Data / Schema / Behavior Abstraction:

Protocol (deployment): Functional "Dialog" Augmentation. Semantics Protocol (Dataflow Message).

Applications: Hypermedia Dataflow Activation (reactive / event driven knowledge based contents). Dataflow layers / levels (Forms / Flows).

Distributed: Consistency. Inference of distributed state. Event sourcing. Trust. Reconciliation.

Connected application sources (backends: EAI / ESB) and declaratively stated application models integrated by ontology matching bidirectionally.

ToDo.

## Features

### Ontology matching

### Augmentation Protocol

### Reactive / Event Driven

# RDF Introduction: Graphs, Triples, Quads

# RDF Quads for Object Graph Representations

# Models

OntResource is the class responsible for aggregating different URIs referring the same entities (Ontology Matching).

The idea is to infer Schema (classes and instance of classes / relations) operating over Data layer. Then, by aggregating Data and Schema, infer Behavior (classes and instances of operations / functions). This Aggregation, together with Alignment and type Activation mechanisms comprehend the Model core Augmentations.

## Model Abstraction

### Models: Layers

What my attempts where about in the beginning was to match different URIs or, for example, database 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.

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.

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 according the hierarchy layer level it corresponds (and declaratively stated in a Model of Meta Resources).  
  
Layers shape is as follow:  
Resource : Functional URI wrapper.

(Context : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);

CSPO Names are according roles (Meta Resource) in the Model. For example: layer Occurrence is parent layer class.  
Each layer abstract instances of its own contexts instances.

Input Layer: (CSPO layer):

(Transaction, someOne, buys, someProduct);

Statement (data layer instance):

Inputs regarding the same context are aggregated into data layer instance.

(Statement, Occurrence, Attribute, Value);  
(transactionStatement, someOne, buys someProduct);

Entity (data layer class):

Aggregated Statement and Occurrence Statement occurrences reified into an Entity along with its Occurrences Attributes.

(Entity, Statement, Occurrence, Attribute);

(someTransaction, transactionStatement, someOne, buys);

Role / Kind (schema layer instance):

Aggregated Entity and Statement Entity occurrences reified into a Role / Kind along with its Statements and Occurrences.

(Role / Kind, Entity, Statement, Occurrence);  
(someBuyer, someTransaction, transactionStatement, someOne);  
  
Class (schema layer class):

Aggregated Role and Entity Role occurrences reified into a Class along with its Entities and Statements.

(Class, Role, Entity, Statement);  
(Person, someBuyer, someTransaction, transactionStatement);  
  
Flow (behavior layer instance):

Aggregated Class and Role Class occurrences reified into a Flow along with its Roles and Entities.

(Flow, Class, Role, Entity);  
(someBuy, Person, someBuyer, someTransaction);  
  
Behavior (behavior layer class):

Aggregated Class and Role Class occurrences reified into a Behavior along with its Classes and Roles.

(Behavior, Flow, Class, Role);  
(Buy, someBuy, Person, someBuyer);

Then, each Model aggregates its Statements in the form (for example):

(Model Impl, Buy, someBuy, Person); Interaction / Meta Model.  
  
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 Meta Model. The act of applying an Augmentation implies one source Message Resource (context layer), one matching Template Resource (input signature) an Augmentation (Interaction functor) a Transform Resource (output signature) and a resulting (set of) Message Resource(s) materialized as further layers instances / Messages to be “parsed” by further corresponding Augmentations of matching Template signatures (dataflow).

### Models: Facets

#### Functional Model Facet:

#### Semantic / Semiotic Model Facet:

#### Dimensional Model Facet:

### Models: Levels

Models have “Ontology” Levels. Levels are Layers (of the Model) which are feed into its input Layer with (instance) Statements aggregated from initial input data (Data Level) aggregated into subsequent layers. Schema Level instances feeds the Model input conforming a Session (context / grammars) ontology Level. Then, behavior Level instances feeds the Model input conforming an Interaction (behavior) ontology Level.

#### Model Source Level (Backend)

Input Statements coming from plain RDF Quads aggregated according Data / Schema / Layers Augmentation(s). Base facts Model Level.

#### Model Session Level

Aggregate Source (Backend) Level Schema layer Statements as Model Session level Data layer input. Reify Schema (roles / grammars).

#### Model Interaction Level

Aggregate Session Level Behavior layer Statements as Model Data level Data layer input. Reify behaviors (context / interactions). Declarative application protocol use case upper ontology levels (Action… Gesture, etc).

## Meta Resources

Meta Resources are used by a Model Meta Model for describing (reified) models. Some of them are:

URI, Resource, Context / Subject / Occurrence / Predicate / Attribute / Object / Value / Statement / Model / Kind / ContextKind / SubjectKind / PredicateKind / ObjectKind / Message / Template / Augmentation /Transform / Class / Metaclass / }Instance

(Augmentation, Template, Mapping, Transform); Meta Resources.

Model declared as Interaction Model Augmentation (matching Messages) in Interaction Model. Flows:

Message

Model (Functor)

Augmentation (Addressable Interaction)

Template (Message)

Mapping (Functor)

Transform (Message)

Model (Functor)

Message

Interaction Model: Model Events (Augmentation).

Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output.

Embeddings: Message Match Event Signature. Tempate matching / Transform rendering. Dataflow.

Embeddings: Meta Model Augmentation, Template, Mapping, Transform Meta Resources (input layer). Meta Model Source, Session, Interaction levels.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

## Meta Model

Meta Model: encode Layers, Contexts, Kind / Roles hierarchies (subject, context, occurrence, roles, atributes, values / metaclass, class, instance relations / meta resources) and Facets using corresponding Facets implementations of base Model Meta Resources.

URI(s);

OntResource; Merged URI(s) wrapper.

OntResource hierarchy: layers statement contexts. Facets DOM, Actor / Role.

Resource (OntResource Context Roles hierarchies Monad wrapper);

Resource (OntResource CSPO / Contexts hierarchies Monad wrapper);

Role (Model CSPO Context Roles hierarchies type classes) : Resource;

Statement (Resource, Resource, Resource, Resource) : Resource;

Kind (Statement\*) : Resource;

Class (Kind\*) : Resource;

Context (Class\*) : Resource;

Hierarchy: class (Object / Value) as superclass Context.

Object: class (extension);

ContextStatement: super class (intention); Context Role.

(Resource, ?, ?, ?);

(Role, Resource, ?, ?);

(Statement, Role, Resource, ?);

(Kind, Statement, Role, Resource); Data (Resource Kind).

(Class, Kind, Statement, Role); Schema (Role Class)

(ContextStatement, Class, Kind, Statement); Interaction (Statement ContextStatement).

State Facet / Layer / Level / Augmentation / Model Resource Mappings. Functional API: Message IO. Mappings.

## Interaction Model

(Augmentation, Template, Mapping, Transform);

Model declared as Interaction Model Augmentation (matching Messages) in Interaction Model. Flows:

Message

Model (Functor)

Augmentation (Addressable Interaction)

Template (Message)

Mapping (Functor)

Transform (Message)

Model (Functor)

Message

Interaction Model: Model Events (Augmentation).

Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output.

Embeddings: Message Match Event Signature. Tempate matching / Transform rendering. Dataflow.

Embeddings: Meta Model Augmentation, Template, Mapping, Transform Meta Resources (input layer). Meta Model Source, Session, Interaction levels.

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

Encoding. Addressing (contents, signatures, contexts). Events publish / subscribe. Dynamic subscriptions / bindings. Subscription, reactive Meta Resource(s). Message flow mechanism: from Model to base layers.

Match Message subjects to Templates, Augmentations and Transforms roles (bound by CK signatures dataflow). Message inputs: Models.

Interaction Model for Encoding / Addressing (Mapping : Event routes) Dataflow metadata.

Augmentation: Described in Interaction Model. Encode Message, Template, Augmentation and Transforms roles (Meta Resources). Data, Session, Interaction Levels (Message, Template, Transform, Augmentation statements declaration realization).

Augmentation. Aggregation Meta Model: Describe layers contexts compositions. Alignment Meta Model: Describe augmented attributes (by kinds clustering). Activation Metamodel: Describe Kinds / Roles activation (by attributes aggregations).

## Dataflow: Mapping

Interaction Model declares Events (Augmentations) which have a functional Mapping between its domain (Template) and range (Transform). An Augmentation Context Kind correspond to this Mapping “signature”. Dataflow binds input Message(s) to domain Template by pattern matching and resolving any input Message references (Addressing).

Outputs are resolved by pattern matching with Transform, Message and existing Model data. Augmentations may play the role of “placeholder” Resource(s) which are bound to context aware Augmentations thus rendering Transforms into Model entities (including Mapping Augmentations themselves).

Model declared as Interaction Model Augmentation (matching Mappings) in Interaction Model. Flows:

Message

Model (Functor)

Augmentation (Addressable Interaction)

Template (Message)

Mapping (Functor)

Transform (Message)

Model (Functor)

Message

Interaction Model: Model Events (Augmentation).

Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output.

# IDs: Encoding / Addressing

Encoding, IDs: magic numbers (MIME types : Context Kinds), metaclass, class, instance, context, CSPO, etc. relations "contextual slots" for IDs. Resource resolution, Operation (primes, encoded lattice, slots context relations) factors in Meta Model relations. Encode order, hierarchies, temporal, causal (reified), containment, etc. relations into IDs encoding. Ontology matching: encoded IDs roles in context aggregation / learning.

Explain URI, Resource, Layers, Model, Kinds, etc. APIs. Meta Resources. Meta Model. Hierarchies. Order. Iteration. Flows.

Model, URIs, Resource, Contexts Functional APIs. Meta Model / Resources encoding. Mappings.

IDs: Addressing / Encoding. Semantic (signature, contents, context) resolvable / discoverable identifiers. Messages CRUD / Invocation semantics. Dialog. Prompts.

Encoding: Cons lists. Trees. Huffman / Prefix codes. RDF List serialization. Meta Resources / Models declarative Encoding, Addressing, Mappings, Transforms (Immutable sequences, dataflow Mapping: Template / Augmentation / Transform functional streams).

Kinds, Signatures. Contents. Contextual metadata. Sets (bitstring cuads). Lattices.

(C (S (P (O, Nil))));

(C2 (C (S (P, Nil)));

## IDs: Encoding / Addressing

Contextual / patterns embedding metadata / resolution. Augmentation occurrences Dataflow sources / context sync.

## IDs: Encoding / Addressing

Contextual / patterns embedding metadata / resolution. Augmentation occurrences Dataflow sources / context sync.

## IDs: Encoding / Addressing

Ontology Matching:

Ontology Matching: IDs, Addressing, Encoding. Functional, Semiotic, Dimensional (Facets). Layers. Levels. Meta Resource / Model. Sets. Value as occurrence of attribute. metaclass / class / instance IDs.

Ontology Matching: Encode: order, iteration, flows, units, relations, events, enums, etc.

## IDs: Encoding / Addressing

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors behavior encoding statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

IDs: Addressing / Encoding. Semantic (signature, contents, context) resolvable / discoverable identifiers.

## Models / Meta Model IDs

Message (Resource Monad wrapper); Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

IDs:

A: OntResource.

B: CSPO Role.

C: Statement : OntResource Occurrence.

D: Kind CSPO Instances.

E: Class : Kind CSPO Classes.

F: ContextStatement : Context Role.

Meta Model:

A: (Resource, ?, ?, ?);

B: (Role, Resource, ?, ?);

C: (Statement, Role, Resource, ?);

D: (Kind, Statement, Role, Resource); Data (Resource Kind).

E: (Class, Kind, Statement, Role); Schema (Role Class)

F:.(ContextStatement, Class, Kind, Statement); Interaction (Statement ContextStatement).

ID: (F (E (D (C (B (A, Nil))))));

Cons lists. Binary Trees. Huffman / Prefix codes. RDF List serialization. Meta Resources / Models declarative statements Encoding, Addressing.

Kinds, Signatures. Contents. Contextual metadata.Lattices. Roles.Sets (bitstring cuads). Definitions (elements). Operations. Rules. Categories. Groups.

Statement: (ID (ID (ID (ID, Nil))));

(C (S (P (O, Nil))));

## Resolution: graph criteria / query

## Sorting / Order / Comparisons

Order / comparisons: tree representation ordered by Context Role class hierarchy, instances hierarchies and aggregation hierarchies. Resources order (IDs). Statements order (Statement IDs). Comparison criteria (choose relevant IDs). OrderedSet, Order by type hierarchies, order by domain / range of functions.

Semantic resolution: Query Resource(s) satisfying “criteria” (i.e.: Object(s) for predicate) IDs by IDs resolution pattern:

Query Resources by role in context.

Query Resources by attributes / values.

Query Resources by identity / type.

# Model Functional APIs

## Resource Monad

Monadic wrapper for which Augmentation (Functor Events) are declared into Interaction Model. Model(s) themselves are Augmrntation(s). Augmentation Statement Context Kind defines Event “signature”: Resource input / output Event domain / range. Output from an Event application (Transform) may feed back Model triggering further events (Dataflow). Augmentation Template, Mapping and Transform may behave as placeholder for Dataflow rendering of Meta Models.

## Reactive / Events (Resource Monad)

Model Resources react to events according Message matching event “signature”.

## Meta Resources

Declare Meta Model Meta Resources as Interaction Model Augmentations, Mappings, Templates and Transforms. Model(s) (Facets) are “root” Augmentations over input Message Statements. Further Dataflow and Meta Model Interaction Resources embeddings shapes Meta Model instances.

## Meta Model

Meta Resources built Model which instantiates Interaction Model statements into layered Facets Models.

## Streams (Context, Kinds, etc.)

Context Kind: Functional stream of Context Statements (Occurrences).

Subject Kind: Functional stream of Subject Statements (Occurrences).

Predicate Kind: Functional stream of Predicate Statements (Occurrences).

Object Kind: Functional stream of Object Statements (Occurrences).

# Messages

Messages: Mappings. Meta Resources / Model Message based Model interactions (Subscriptions / Mappings). Messages: Message semantics (Augmentation: Verbs, CRUD, Behavior) according Message structure / pattern (dialog / prompts).

Messages: Dataflow Template matches signatures (Session level, enrichs Message with Model / Dialog prompts / contents). Augmentation Functor applied over Message contents (Interaction level). Transform matching output signature emits (Session level, populated / prompts) output Message.

Messages: Dataflow. Subscriptions. Reactive Model. Dynamic subscriptions / bindings. Events publish / subscribe between Model Resource. Mappings.

Mappings: Declarative IO signatures: Context Kinds Templates / Transforms. Subscriptions / routes. Dataflow.

Messages: Mappings. Meta Resources / Model Message based Model interactions (Mappings : Subscriptions).

Messages: Message semantics (Augmentation: Verbs, CRUD, Behavior) according Message structure / pattern (dialog / prompts).

Messages: Dataflow Template matches signatures (Session level, enrichs Message with Model / Dialog prompts / content alignments). Augmentation Functor applied over Message contents (Interaction level). Transform matching output signature emits (Session level, populated / prompts) output Message.

Messages: Dataflow. Subscriptions. Reactive Model. Dynamic subscriptions / bindings. Events publish / subscribe between Model Resource. Mappings.

## Message Monad

Encode Model Events Interactions (Augmentation) inputs (Template) and outputs (Transform) wrapping corresponding Resource(s).

## Reactive / Events (Message Monad)

(Augmentation, Template, Mapping, Transform);

Model declared as Interaction Model Augmentation (matching Messages) in Interaction Model. Flows:

Message

Model (Functor)

Augmentation (Addressable Interaction)

Template (Message)

Mapping (Functor)

Transform (Message)

Model (Functor)

Message

Interaction Model: Model Events (Augmentation).

Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output.

## Model Functional APIs

Invoke Augmentation over Resource Message matching Event signature.

## Persistence

Messages: Events IO / Persistence: Saga Activation / Passivation populating Node local Quad store / persisting peers via DIDs (Distributed IDs) / Semantic (resolvable / discoverable) identifiers.

Interaction Model: aggregated Meta Model interactions (performed / inferred / possible) declared Models events (saga pattern).

Augmentations: Interaction Model Mappings execution / persistence / retrieval. Reactive model via representation of IDs: Mappings (signatures) dataflow inferred Augmentations.

# Augmentation

Interaction Model Event. Matches Message signature (domain Template / range Transform) performing Mapping. Dataflow: Transform output matches another Event signature. Embedding: OntResource augmented with new referenced aligned / matched Model entity.

Augmentations defined as declarative Mappings in Interaction Model encoding Context (layer) inputs matching signatures and augments current / previous layer emmiting mapping transforms.

Augmentation:

Functors: Augmentation declaration: Meta Model definitions (Context class / instances). Message: dataflow matches Template signatures: interactions. Apply Augmentation Functors over Message contents (interactions enrich Message with Models contents: ontology matching / Levels / Facets). Materialize / emit dialog / prompts Message (enrich Message from Models / reactive IO events).

Augmentation: Context / Functors. Message Resource(s) / Meta Resource(s) (nested / wrapped) elements determines flow Template Transform results / behaviors (CRUD, Functor invocations). Message IO performs Augmentations. Ontology levels resolution (Templates / Transforms / Augmentatiom levels: matching patterns / dialog prompts in Ontology levels).

## Reactive / Events (Functors)

Augmentations: matching Events Functors aggregate / align / activate (classify) sources of ontology matched data / schema / behavior enabling semantic layers interoperation.

Aggregation: Infer input data streams data, schema, behavior class / instance context layers.

Alignment: Infer layer missing / deducible attributes and values.

Activation: Infer layer CSPO Kind / Roles. Basic type system.

Aggregation (Augmentation): Apply each Context (layer) Functor on inputs (from input layer) and emits Transform, matching corresponding (next) layer. Next layer Context and SPO according functional mapping declared by Meta Resource types on augmented layer.

Alignment (Augmentation): ToDo.

Activation (Augmentation): ToDo.

Order: OrderedSet / class hierarchy / domain / range.

## Interaction Model

(Augmentation, Template, Mapping, Transform);

## Message

Augmentation Event input (Template) / output (Transform) declaration / instance (Mapping).

## Augmentation

## Template

## Mapping

Mappings / Augmentation Context Kind.

## Transform

Augmentation Mapping range declaration / result instance (Transform Message).

# Dataflow

(Augmentation, Template, Mapping, Transform);

## Reactive / Events (Functors)

Augmentations: matching Events Functors aggregate / align / activate (classify) sources of ontology matched data / schema / behavior enabling semantic layers interoperation.

## Interaction Model

(Augmentation, Template, Mapping, Transform);

## Messages

## Augmentation

## Templates

## Mappings

## Transforms

# Ontology Matching

Ontology Matching: IDs, Addressing, Encoding. Functional, Semiotic, Dimensional (Facets). Layers. Levels. Meta Resource / Model. Sets. Value as occurrence of attribute. metaclass / class / instance IDs.

Ontology Matching: Encode: order, iteration, flows, units, relations, events, enums, etc.

## Ontology Merge

Ontology matching. Dataflow: sort statements (type hierarchy, domain / range). Units. Equivalences. Distances / events (order). Services (Augmentation / Context Functors Meta Model mappings / transforms).

Explain ontology matching: data, schema, behavior alignments. Layers. Levels. Facets. Meta Resources / Model. IDs, Encoding / Addressing.

Encoding. Functional, Semiotic, Dimensional (Facets). Layers. Levels. Meta Resource / Model. Sets. Value as occurrence of attribute. metaclass / class / instance IDs.

Functional / Semiotic / Dimensional layers / levels examples / alignments.

Model, URIs, Resource, Contexts Functional APIs. Meta Model / Resources encoding. Mappings.

IDs: Addressing / Encoding. Semantic (signature, contents, context) resolvable / discoverable identifiers.

Ontology Matching: IDs, Addressing, Encoding. Functional, Semiotic, Dimensional (Facets). Layers. Levels. Meta Resource / Model. Sets. Value as occurrence of attribute. metaclass / class / instance IDs.

Ontology Matching: Encode: order, iteration, flows, units, relations, events, enums, etc.

## Alignments (data / schema / behavior).

Ontology Matching (data, schema, behavior: reified operations as schema / data).

Data alignment, Schema alignment, Behavior alignment,

# Implementation

Persistence:

XML / XSL. Event bus (encoding / discovery). Addressing (node / model / ontology levels, topics / queues). Implementation: Spring / Vert.x. Persistence: Event Sourcing. Topic / Subject wrapping (DIDs) Saga (Semantic IDs) Messaging pattern.

Index, Registry, Naming Hypermedia Service Beans. Backend, Session, Interaction Levels: Functional Service stream APIs Beans. Functional Service URIs. DOM (Dynamic Object Model) OGM (Object Graph Mapping). Beans API. JAF (JavaBeans Activation Framework). REST / Client OO APIs. Service Bean.

Core protocol. HATEOAS protocol. Forms / Flows. APIs (DDD). Forms / Flows: HATEOAS HAL. MVC. REST. Meta DCI Models protocol based.

# Client APIs / Connectors

## Services

Hypermedia APIs: Augment, Extend, Declare. REST. Extended Content Type signatures Activation / Dataflow. DCI Activation: DOM OGM. REST. API Client. Services. Wiki like abstract representation / protocol. Template rendering. Services. DAV. Service (backend / plugins) reactive URIs (example: JDBC plugin).

# Deployment

Protocol plugins (Protocol Service) Connectors. Runtime. Core Services. Endpoints. Dataflow.

JavaScript (browser) / NodeJS Core.

Endpoints. Connectors (OData, HAL, OGM, Spring, ServiceMix / Fuse).

Browser (JavaScript) / NodeJS / Connectors.

Applications. Use Cases. Levels. Gestures / Actions (Services / UX). Rendering (REST APIs / Dynamic UX). Deployable entity: Node. Publish / Subscribe signatures (interface).

**FCA / Lattices**

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.

FCA Context (tables): Context (Relation, Kind, Role, Statement, Context, etc.) matrices w./ corresponding CSPO x CSPO types axes. Types intersections determine cell type (as in example tables shown). Values intersections are instances of corresponding types. Example:

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

Thus, each layer context statements are used to build a matrix of CSPO x CSPO of its types and (scaled) values. In this manner (aggregating matrices / FCA contexts), SPO Resource occurrences in matrices axes SPOs / context layers statements SPOs are reified in statements for which the occurring SPO Resource is the statement Context and its SPOs are resolved according its SPO position in the original Context. This statements form the matrix rows and columns SPOs according some aggregation layout.

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).

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

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.

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 “generic” form of the table is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Context | Occurrence | Attribute | Value |
| Context | Context | Occurrence | Attribute | Value |
| Occurrence | Occurrence | Context | Value | Attribute |
| Attribute | Attribute | Value | Context | Occurrence |
| Value | Value | Attribute | Occurrence | Context |

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.

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).

If Context types / instances identifiers are sequential in form, a mapping (hashing) could be done in a bitstring of the length of scaled attributes (columns) having a 1 in the corresponding attribute for a given object. Attributes also may be rendered as a sequence of prime numbers being an object extension the product of its attribute primes.

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.

Encoding: Layout rules. Context layers, use layout to aggregate occurrences matrix. Use aggregated occurrences matrix to retrieve original matrix.

Encoding: Layout. Layers. Aggregation: upper layers / lower layers. Encode Augmentations. Browse. Transforms. Reified Model. DCI.

Layer declarations: TBD (Context, Occurence, Attribute, Value) reified types / data.

Resource matching (reified / data): context / occurrence rules. Context population. Types / Functional / Data Models.

Layer downwards: Layer for which Subject is Context.

Layer upwards: Layer for which Context is Subject.

Facets: Concept hierarchies common attributes. Types: reified model objects instances: contexts / layers / aggregations. Values. Parsing (case match): resolve if an (scaled) type / value object corresponds to an (scaled) type / value attribute and which is its intetsection type / value (grammars / signatures / aggregation / dataflow). TBD.

Facets: cells Context type: from corresponding Kinds for reified CSPO roles of SPO axes intersections (type intension). Context values: CSPO Resources of Kind type extension.

CSPO / Kinds types / values (FCA scaled types / values objects / attributes intersections): Multiset (OrderedSet) encoding of CSPO Statements and Kinds as bitstring quad (CSPO segments) for corresponding sets / elements: C, S, P, O Resources arranged in a three SPO sets distribution (diagram) with SP intersection for Object Kinds, PO intersection for Subject Kinds and OS intersection for Predicate Kinds (Kinds being referencing SPOs in their corresponding parts and aggregating Kind values in their Kind type part).

Kinds layouts:

(S, P): OK; (P, O): SK, (O, S): PK; (Idem, Idem): Contexts, i.e.: Relations of equivalent Context signatures.

CSPO Sets Context (data layer, context / interaction hierarchies reification) population: parsing over RDF quad statements IO. Create / aggregate quad bitstring Resource / Kind identifiers. Functional query / browse / transform. Populate FCA contexts / lattices and calculate cells / dataflow (sets streams).

Populate Contexts (Sets, FCA, layers):

Resources, Statements: inputs.

Role (CSPO Context hierarchies reified roles), Statement, Resource, Attribute. From sets aggregation.

Kind, Role, Statement, Resource. From sets Kinds aggregation.

Relation, Kind, Role, Statement. From sets reified Kinds / Context / Statements aggregation (Kinds domain / range).

Sets Layout and encoding bitstring mask format:

