Products And Services Community Exchange Network

Resource Oriented Knowledge Computing (ROKC)

Features: Mision / Vision

Distributed Applications and Services Knowledge and Behavior Integration.

Data, Schema and Behavior Alignment. Virtualization (services).

Alignments:

Aggregation (clustering).

Alignment (regression).

Activation (classification).

Semiotic Context: (Context, Object, Sign, Value).

Context, Occurrence, Attribute, Value.

Ontology Matching: Metaclass, Class, Instance, Ocurrence (data / schema / behavior).

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 invocations). Domains Translation. 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.

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 and applications. 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 / editable (JCR).

Objective: 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.).

Integration of addressable resources. Reactive I/O (sync backends). Content type driven semantic augmentation / annotations (alignments).

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: Subject (image URI / resource : source), Predicate (detection service / index service), Object (detection / search results endpoint / placeholder : destination). Grammars: Predicate Kind (face / search recognition signature) from Subject (faces images / names) / Object (face classes / subjects) Kinds. Kind model layers. Models definition: data (Statement, Entity), schema / context (Role, Class), interactions / behavior (Flow, Behavior).

Semantically annotated content types: image/png;face, text/xml;faceImgCoords. RDF schemas describing content, attributes, links in context / target roles. Content types: labels (schemas).

Message: Context Model API. Input statements: Model Grammar. Augmented IO by interaction transforms of applied matching Message with model statements inputs. Context of core models instances. API.

Base core services URIs (index, naming, registry). URI subclasses implementing / wrapping state for Resource monads offering protocols / addressing / content types / representations facades for services: DBs, WS (REST, SOAP, SPARQL), ML (predictions), etc.

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.

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.

Model, Contexts, Interactions IO:

Resource, Statement, Kind, Message, Transform.

Subscription, Subscriber, Producer, Consumer, Processor.

Model: aggregated resource statements model.

Context: aggregated model kinds (grammar statements model).

Interaction: aggregated model / context / dialogs bindings model.

Aggregation: layers. Parameterized Resource<C, S, P, O> : CSPO : URIs hierarchy.

Materialized interactions re-populate model and context (Augmentation). Browse context model: kinds and grammar known statement "templates" (by kinds hierarchy layers aggregation) navigation for discovery of domain messages resource kinds.

Model, Context, Interaction IO: Message. Nested CSPO contexts quad, CSPO resources (plain URIs, kinds, nested contexts). Wildcards, variables, placeholder, null values: Message structure defines CRUD behavior. Dimensional / order resolution: common upper super types.

Message: Resource model hierarchy parent class (monad of plain URI, parameterized resources). Resource set specification. Any Resource is a Message, specifying a potential set of other Message (Resource) in a model (layers).

Resource : Message. Resource resolution: known URIs, known resource kinds bindings, dialog (resource set specification) recursively. Interaction model (dialog resolved resources set). Wildcards, variables, placeholder, null values: Message structure defines CRUD behavior.

Resource monad of URIs or Message monad of Resource?

**Behavior Model (Dimensional annotations)**

Dimensional Model layers population / augmentation. Purpose modelling. Dimensional Concepts.

Order layers statements. Hierarchies (contexts / kinds). Parent / child relationships (steps). Order type relationships: husband: single / marriage / married.

(Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.  
(Measure, Value, Previous, Distance);  
(Unit, Measure, Value, Previous);  
(Dimension, Unit, Measure, Value);  
(Concept, Dimension, Unit, Measure);  
(Resource, Concept, Dimension, Unit);  
(Statement, Resource, Concept, Dimension);

Populate / align / annotate models with dimensional data. Model input: statements (model resources). Model specification: augment, sort  
statements. Model specification: specialization of base model layers. Resolve resolution statements order.

Dimensional input set model specificatíon (from Statement layer, ordered SPOs: order criteria, comparisons. Kinds / class / occurrence / instance  
order criteria?).

Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements layers).

Value -> distance(prev, next); ordering;

Assert knowledge: 1h -> 60min;

dom-lun-mar-mie-jue-vie-sab (orders);

1mt -> 100cm;

etc.

Comparison / order: Alignments (prev, curr, next asserted knowledge). Next hour, location, city, country, next distance at next time at current speed. Event sourcing / tracking: married -> marriage occurred.

Sort: cause / effect, temporal, etc. Messages align, functional map, fold, etc. Primitives. Encode layered statements ordering. Complement / supplement concepts definitions.

Events metamodel (TBD):

(Object, State, Axis, Type)  
(State, Axis, Type, Event)  
(Axis, Type, Event, Event)  
(Type, Event, Event, Event)  
(Event, Event, Event, Event)

Dimensional alignment / aggregation layers (lower resource alignment layers):

(Value, Distance, Prev, Next : in Units); (Measure, Value...) (Unit, Measure, Value,...); (Resource, Unit, Measure, Value); Marriage event example.

Model Contexts: Meta Resources / Contexts hierarchies. Models:

Data: Source / Interaction, Schema: Encoding / Grammar, Behavior: Dimensional / Measures (marriage).

Example:

(Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.

Order layers statements. Hierarchies (contexts / kinds). Parent / child relationships (steps). Order type relationships: husband: single / marriage / married.

(Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.  
(Measure, Value, Previous, Distance);  
(Unit, Measure, Value, Previous);  
(Dimension, Unit, Measure, Value);  
(Concept, Dimension, Unit, Measure);  
(Resource, Concept, Dimension, Unit);  
(Statement, Resource, Concept, Dimension);

Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements layers).

Value -> distance(prev, next); ordering;

Assert knowledge: 1h -> 60min;

dom-lun-mar-mie-jue-vie-sab (orders);

1mt -> 100cm;

etc.

Comparison / order: Alignments (prev, curr, next asserted knowledge). Next hour, location, city, country, next distance at next time at current speed. Event sourcing / tracking: married -> marriage occurred.

Sort: cause / effect, temporal, etc. Messages align, functional map, fold, etc. Primitives. Encode layered statements ordering. Complement / supplement concepts definitions.

Encoding / Dimensional example: role in context. X is Y for Z in W.

(W (Z (X (Y))));

(Marriage (Role (Man (Husband))));

(Hour (Minute (1 (60))));

Encoding, Dimensional, Meta Model. Units. Events. Order. Relations. Comparison. Input layers. Augmentation.

**Forms / Flows**

Dialog arguments resolutions example: higher layer Resource / Message request / invocation instantiates in Interaction Transform context corresponding lower layer graph statements to be “populated” to fulfill request. Message IO of “forms” (Messages) inter-peers (originating peer  
acting as “server”) for initial requested peer to “ask” for form elements to be populated (interaction context “dialogs”). Resolution may propagate to other peers (content aware addressing dataflow routes dispatch: P2P resources address encodings, matching forms models requests). Nested interactions.

Explain messages (resource resolution). Grammar. Match model Resource(s). Compound nested CSPO statement contexts defines result behaviors. Message CSPO contexts may define create, retrieve, update or delete operations (passing 'null' for example for resource / statement to be deleted).

Explain transforms (message application). Transform: Resource stream result of Message application over resolved Resource(s)). Input statements: Message(s) / Resource(s) (from input message or to be populated or populated in dialog) and "goal" Message / Resource aggregating a model from Resource MetaGraph with Message / Resource bindings.

**Activation:**

Available navigational browse / transforms (Forms / Flows).

CQRS. Event Sourcing. Protocol / Encoding: hierarchical contexts dialog (runat) prompt / pick / select (roles).

**Layers Aggregation:**

Ontology Matching: Dimensional metaclass / class / instance attributes relationships in axes / contexts / roles evaluations. Resolve equivalence of dimensions, units, measures, values via aggregation of value occurrences. Resolve value kinds by occurrences of Semiotic (grammar) model. Contexts (CSPOs) Attribute / Value "clustering". Sets specifications. Dimensional order relations.

Distributed Inference (dimensional / matching / predictions). Distributed consistency. Event sourcing (Dialog, CQRS Protocol). HATEOAS distributed / lazy reconciliation: hashing of source / destination (hierarchical contextual model quads hashing) state in interactions tokens. Prompts. Distributed Alignment (until reach of goal synchronization states). Distributed state / IDs.

Comparable: parent / child relations (SortedSet). Model sets / dimensional axes alignments.

Semiotic: syntax, semantics (grammar), pragmatics. Verbs: action, passion, state (for / due to action / passion). Models. Dimensional (axis / order) relations. Alignments. (Sorted) Set relations: hierarchical encoding. Verb roles (CSPO / Kinds sets).

Example aggregation: candy (type), red (color attribute), strawberry (flavor attribute).

Encode IDs: Context Kind, upper (meta) Resources (levels / layers). Resource contents / contexts (identify by occurrences in roles in other contexts, Meta Resources, layers class, metaclass, instance). Compose IDs (hierarchical graph properties encoded string) from outer to inner resources (Context, Kind, Occurrence, Role, Resource). "Operable" IDs (ClassIDs / InstanceIDs: Meta Model reifications / occurrences).

Message parsing (Template, Message Context) matches Form / Flow Augmentation Mappings signatures.

(Template, Context, Attribute, Value); Value as Context: hierarchical models. Same attributes: types / collections.

Semiotic (encodings):

(Context, Sign, Concept, Object);

Object as Sign: Object properties (Concepts).

Semiotic encoding: encode Meta Model (syntax facet) / Functional (Meta Model encoded pragmatic facet) / Dimensional (Meta Model encoded semantic facet) data / reference model (by contexts / upper alignments / aggregation).

FCA. ML. Map Reduce encoding inputs, grammar templates context mappings. Emit Semiotic reference / data model parent / child properties encoding (Context / SPOs, Context:Subject / POs, etc.).

(Context, Sign, Concept, Object);

Resource Monad / Message Functor (Contexts hierarchy parent). Resource<Context>. Events bindings. Message / Augmentation event declarations / instances.

Functional: Entities in different Models in Kind, Class, Flow, Behavior.

Dimensional: Measures in different Dimensions in Units, Values.

Value, Previous, Distance / Event, Next. Order (in axis):

Siblings: previous / next (Semiotic containment relationship / roles). To do.

Hierarchies: parent / children (Semiotic containment relationship / roles). To do.

Semiotic model: reification / primitives / upper ontology (meta model, inferred / aggregated). Levels: syntax / grammars, semantics, pragmatics. To do.

Use Case: Goals App

Goals App: purpose / goals / domain driven syndication of integrated business / social / cloud application features. User / Groups / Roles Purpose(s), Goal(s), Task(s) "intelligent" tracking oriented focus providing an abstraction and integration layer of players process flows / interactions and players process assets management and semantic orchestration.

Goals App: Semantically annotated gestures / interactions (contexts, purposes messages / interactions / resources / content). Subject context occurrence role attributes values (metaclass, class, instance, occurrences).

Goals App: API Facade for rendering aggregated data roles in contexts interactions topics / subjects assets (conceptual domain contexts axis / state views / activations: Forms / Flows). Example: domain declared Customer (actor / role), Product, Order, Purchase, Invoice, etc. topics / subjects assets rendered in contexts (Sales Report, Expenses Report, etc. embedded / linked dashboards). Wizards.

Goals App: Browse / search / activate: history / relations / referrer context / interaction / gestures roles traceability / (dialogs). Gestures / interactions (actor / asset, actor / actor). Wizards.

Goals App: Hypermedia contents APIs (embedded / embeddable resources: Semantic contextual Wiki / Apache Stanbol / CMS: hypermedia augmentation, knowledge / behavior maps). Integration: augmentation / sync backends / apps. Extension: services / APIs. Annotate / augment link content. DAV protocol (integration / extension facades).

Low level Resource / Message / Context model / layers API. REST. Render DOM Context / OGM Domain (model) instances: Restful Objects / Apache Isis / HAL / GraphQL (meta / domains models endpoints) like APIs. Forms / Flows MVC / DCI APIs (connectors / clients / adapters). Wiki APIs: Knowledge discovery. Assistants.

Languaje levels: Use, mention. Pronouns. Verbs (action, passion, state).

Sign, Interpreter, Concept, Object: relations (syntax, semantics, pragmatics).

Model layers:

OntResource: Resolves reified aligned / matched aggregated Resources.

Predicate: 'kind', aggregates roles attributes / values. Grammar.

(OntResource, OntResource, OntResource, OntResource);

(Predicate, OntResource, OntResource, OntResource); For a Predicate occurrence, attributes / values.

(Message, Predicate, OntResource, OntResource); For a Message Predicate occurrence, possible attributes / values. Functor declaration.

(Context, Message, Predicate, OntResource); Occurrence (object) for a Context (interpreter) Message (sign) Predicate (concept). Adapter.

(Transform, Context, Message, Predicate);

(Mapping, Transform, Context, Message);

(Template, Mapping, Transform, Context);

(Augmentation, Template, Mapping, Transform);

(Resource, Augmentation, Template, Mapping); Type Functor Augmentation instance.

(Role, Resource, Augmentation, Template); Role Functor Augmentation instance.

(Statement, Role, Resource, Augmentation); Augmentation of which Statement is result of. Alignment Functor Augmentation instance.

(Model, Statement, Role, Resource);

(Entity, Model, Statement, Role); Model (Backends) aligned entities.

(Kind, Entity, Model, Statement);

(Class, Kind, Entity, Model);

(Flow, Class, Kind, Entity);

(Behavior, Flow, Class, Kind); Statement, proposition.

(Value, Behavior, Flow, Class); Value on which Behavior occurrence holds.

(Unit, Value, Behavior, Flow);

(Dimension, Unit, Value, Behavior);

(Measure, Dimension, Unit, Value); Truth values. Equivalent Measure(s), comparisons (order / hierarchies). Measure Dimension attributes / values.

Parsing: extract propositions, knowledge assertions (in a domain ontology). Assert propositions links, order, concepts relations (between domains). Link Grammar. ISO TMDM / TMRM.

Parsing: extract prescriptions, knowledge rules (in domain ontology concepts relations: causal, requirements, etc.). StratML.

Parsing: infer possible statements propositions / prescriptions productions. Link Grammar Disjuncts. Embeddings.

Functors: model layers aggregations declarations / instances. Type, Role, Alignment levels. Domain / range: CSPO contexts (Template, Transform signatures). Transform: Mapping Message. Hypermedia events dataflows triggered functors (signature bindings).

Message: Functor Declaration. (events / grammar: protocols).

Augmentation: Functor Instance.

Functors functional declaration: (((a: O, b: S), c: P) d: C);

(dado rango y alcance, universo: U de una relación: P, inferir dominio y codominio, campo: C).

From Object (O) extension /instances to Context (C) intension / class. Incrementally render type, role, occurrence context layers.

Type functor: contexts stream. (((Mapping, Augmentation), Template), Resource). Context layer class / instances.

Role functor: type contexts occurrences stream. (((Template, Resource), Augmentation), Role). Type Context layer class occurrence (Subject) in aggregated context layers.

Alignment functor: type occurrence attributes / values in contexts interactions stream. (((Augmentation, Role), Resource), Statement). Type Subject occurrence attributes / values (statements / augmentation "kinds").

Types / Categories: Resource Monad.

Navigation (TMRM, key / value) functors:

Keys.

Remote keys.

Local values.

Key / value proxies.

Value proxies.

Constraints.

Merge.

Legends (constraints / Mappings Augmentations).

Path languages.

Path expressions.

Encoding. Model layers:

Levels: reify layers from bottom up through contexts hierarchy superclass contexts relationship. Message, Type, OntResource, etc.

Resource context: Resource monad CSPO Form. From Message / Type OntResource / Resource value mappings (functors / navigation).

OntResource context: (Resource, Occurrence, Attribute, Value) Form. Aggregated from Resource context.

(Resource, Resource, Resource, Resource);

(OntResource, Resource, Resource, Resource);

Values. Sets. Equivalences / matching assertions. In Context occurrences, matching attributes / values (recursion to contexts / occurrences).

(Type, OntResource, Resource, Resource);

Labels. Type (C: key) occurrence (S: value), value occurrence (P) aggregated attributes (O).

(Message, Type, OntResource, Resource);

For a Message (proxy / subject context), Type (sign / key / label), OntResource (concept: P), Resource (object / value). Functor declarations. TMRM navigation / paths.

(Context, Message, Type, OntResource);

Context (key / value interpreter / map: connectors legends), Message (sign / proxy / subject), Type (concept / key / label), OntResource (value). Key / Value Adapter. TMRM.

(Transform, Context, Message, Type);

(Mapping, Transform, Context, Message);

(Template, Mapping, Transform, Context);

(Augmentation, Template, Mapping, Transform);

(Resource, Augmentation, Template, Mapping);

Type Functor Augmentation instance.

(Role, Resource, Augmentation, Template);

Role Functor Augmentation instance. Role: CSPO Resource role type in occurrence / context.

(Statement, Role, Resource, Augmentation);

Augmentation of which Statement is result of. Alignment Functor Augmentation instance.

(Model, Statement, Role, Resource);

(Entity, Model, Statement, Role);

Model (Backends) aligned entities.

(Kind, Entity, Model, Statement);

(Class, Kind, Entity, Model);

(Flow, Class, Kind, Entity);

(Behavior, Flow, Class, Kind);

Statements: propositions, prescriptions / rules / productions. DCI / Link Grammar. Entities Statement occurrences: contexts / interactions / proposition Class / Kind roles (satisfaction / rules).

(Value, Behavior, Flow, Class);

Value on which Behavior occurrence holds.

(Unit, Value, Behavior, Flow);

(Dimension, Unit, Value, Behavior);

(Measure, Dimension, Unit, Value);

Truth values. Equivalent Measure(s), comparisons (order / hierarchies). Measure Dimension attributes / values.

Ontology matching: sets, singletons / equivalence classes merge. Encoding: Sets CSPO Contexts specification (sets quad encoding).

Proof of concept: Relational inductive biases, deep learning, and graph networks. Deep Graph Infomax. Train model to extract (augmented) knowledge from training set encoded models. Test output in new encoded examples / different ontologies / domains.

Encoding of types, instances and behavior flows along with the data and information needed to infer knowledge necessary to understand encoded models (meta models).

Model (update / draft). TBD.

Backend: Kernel. Models, Services / Adapters "service" URLs (naming schemes). Messages. Interfaces signatures / content types. JMX, OSGi, CDI, Spring, MQ / Camel (events / bus). Reactive facade for protocol endpoint / service interactions. Dialog (Form / Flow) interaction contexts.

Pluggable "bundles" adapter implementing APIs (i.e.: index, naming, registry services) for each (declarative / functional / reactive) Model / Service / Adapter bundles. HAL (Augmentation / Legends), ActivityStreams, StratML, LoD, Solid Adapters.

Context hierarchies: categories of wrapped subject occurrences. Statement Context monad (category) for Subject occurrences.

Context category instance identity: Employee Kind, Work Behavior, etc. Context statement attribute / value aggregation. Key / value abstraction scoped in category type / instance.

Functors: Layers (Context categories) aggregation / transforms. Augmentations.

Augmentations: Categories Aggregation (types), Alignment (contexts), Activation (roles / matching in interactions).

Locators (keys): (metaclass, class, instance, occurrence);

Metaclass: Transform OntResource

Class: Message Augmentation

Instance: Statement Role

Occurrence: Flow Kind

Locators (keys): local / remote keys. Navigation. Paths. Legends. Signatures (streams).

Monadic  (locators / signature streams based) category wrappers Functors:

M<T>::flatMap(functor(T) : U) : M<U>;

Dado rango y alcance, universo: U de una relación R, inferir dominio y codominio, campo: C.

From Object (O) extension /instances to Context (C) intension / class.

Aggregation (types) functor signatures: stream.

Alignment (contexts) functor signatures: stream.

Activation (roles / matching in interactions) functor signatures: stream.

Encoding. Model layers:

URIResource context: CSPO form. RESTful / HAL monad: HTTP category functors.

OntResource context: (Resource, Occurrence, Attribute, Value) form. Aligned (matched) URIResource(s).

Resource context: OntResource (aligned / matched URIResource) occurrences in reified Role in Statement.

(URIResource, URIResource, URIResource, URIResource);

(OntResource, URIResource, URIResource, URIResource);

Aggregated URIResource OntResource attributes / values (recursion to attributes / values OntResource).

(Transform, OntResource, URIResource, URIResource);

(Mapping, Transform, OntResource, URIResource);

(Template, Mapping, Transform, OntResource);

(Augmentation, Template, Mapping, Transform);

(Message, Augmentation, Template, Mapping);

(Context, Message, Augmentation, Template);

Model.

(Resource, Context, Message, Augmentation);

(Role, Resource, Context, Message);

Reified CSPO / Resource, Occurrence, Attribute, Value Resource role types in Resource occurrence / context.

(Statement, Role, Resource, Context);

(Entity, Statement, Role, Resource);

Aggregated "subject" occurrences of Resource in Role in Statement(s).

(Class, Entity, Statement, Role);

Aggregated Entity Role occurrences type (attributes).

(Kind, Class, Entity, Statement);

Aggregated kinds / roles ("interfaces") of Class occurrences.

(Flow, Kind, Class, Entity);

Action "instance". Entity of Class performs role (Kind) of Behavior Flow.

(Behavior, Flow, Class, Kind);

Action "class". Statements: propositions, prescriptions, rules, productions. DCI / Link Grammar. Context satisfaction (rules).

(Measure, Behavior, Flow, Class);

(Unit, Measure, Behavior, Flow);

(Dimension, Unit, Measure, Behavior);

Order.

Encoding:

Radix, Digit, Position. SPO: radix:digit. Radix: resource count. Digit: resource radix index. Position: context hierarchy aggregation. Normalized radix encoding: metaclass, class, instance, occurrence context metadata operable IDs. Example: Base 2 normalization, concepts lattice bitstring (routing tables), boolean operations metadata extraction / traversal (contexts masks). IPv6 addresses. Masks (contexts), routes. Address resolution resolves recursively expanded (CSPO hierarchy contexts / occurrences / attributes / values) resources representations (graph statements members nested browseable addresses of ID contexts). Addresses representing behaviors (HATEOAS: browse representation links / addresses renders state / operations, i.e.: address plus according referrer context and representation addresses links states CRUD operations).

Normalization: contextual form IDs. Internal four sides polygon angles for each CSPO addressable IDs. Embeed operable metadata in CSPO contextual IDs. Vector space model like.

CAM (Ternary bitstring). Routes. CSPO Functional Mappings (contextual / occurrences vector IDs).

Addresses browse state transforms: navigation renders context layers statements transforms with contextual browsing state (IDs, referrer, contextual / occurrences vector IDs, metadata) as parameters. Contexts navigation (Forms, Flows) as functors. Navigation state transforms: possible activations context / argument / attributes / values. Dialog / prompts (argument resources navigation

state transforms).

Model Encoding: Property graph. Properties (prefix codes, key / value, reification). Sets, groups, categories. Functors applications: Transforms as graph navigation / browse. Template Message parsing (grammar, verbs, state flow). Contextual Quad Context ID: ID according occurrence in Statement context (normalized forms). Occurrence Context IDs indices / mappings.

JAF / Naming / Registry (HATEOAS Forms / Flows navigation / states): DCI / MVC Engine.

Signatures: CSPO Context Kind (Statement Subject Kind + Object Kind). Context Dataflow domain / range (Context as reactive streams producer / consumer).

**3.: RDF: Introduction**

**4.: RDF Quads / Object Mapping (DOM / OGM)**

As RDF Quads encodes four URI values (CSPO Statement) an Object - RDF Quad elemental mapping could be implemented regarding an RDF Quad Statement CSPO as follows:

(C: Context, S: Occurrence, P: Attribute, O: Value);

where Context (C) is the URI of an Object Class identifier, Occurrence (S) is the URI of an Object Class Instance identifier and, aggregating same Class / Instance pairs, Attribute (P) and Value (O) are, respectively, Class Instance member (name, domain / range) and values for the aggregated (S) Object of Class (C).

Contexts. Occurrences, Attributes, Values: Roles of Meta Resource(s) in contexts.

Subject in Statement has Predicate and Object Attribute / Value (roles).

Predicate in Statement has Subject and Object Attribute / Value (roles).

Object in Statement has Subject and Predicate Attribute / Value (roles).

Value as Occurrence of Attribute in Attribute Occurrence Context.

Context Kind (signature): Subject Kind and Object Kind Attribute / Value (roles).

Subject / Occurrence / Context / Role : Attribute, Value. Concepts. Semiotic Metamodel. Dimensional Encoding: each type as each (pair) kind. Pairs (tags / facets).

Meta Model: Layers Resource relations:

Instance, class, metaclass, occurrence, role. DOM, Actor / Context / Role.

Layer Context: Statement class. Aggregates same Context Statement(s). Next layer metaclass (Occurrence)..

Layer Occurrence: Statement Context metaclass. Aggregates same Context / Occurrence Statement(s). Previous layer context.

Layer Attribute: Statement Context Ocurrence Attribute (occurrence). Previous layer Occurrence.

Layer Value: Statement Context Occurrence Attribute Value (role). Previous layer Attribute.

Layer Aggregation begins with Model initial Statement having a new Context (class) “pushing” previous CSPO right, being the new class the new layer Context and CSP becoming SPO:

(C, S, P, O) : (N, C, S, P).

Functional / Object Oriented Resource API (Model, Statement, Semiotic, Dimensional layers, Meta Resources).

Kinds (Application):

Kind: Basic type inference. Applied over layers CSPO during Activation Augmentation. An Occurrence Attributes / Values, aggregated for its URI and Context, determines Kind "members" (Attribute) and Kind instance member values (Value).

Super Kind / sub Kind hierarchy relationship is given by a set of Kind Attributes being super set / sub set of each other.

Examples.

SubjectKind (meta Resource): For a given URI occurring as Subject (Occurrence) across a set of Statements (Contexts), its aggregated Predicates (Attributes) defines its "Kind" and its Attribute values determines the given Kind instance "members" values.

ObjectKind (meta Resource): for a given URI occurring as Object (Value) over a set of Statements, Subject (Kind Attribute), Predicate (Kind Value).

PredicateKind (meta Resource): for a given URI occurring as Predicate over a set of Statements, Object (Kind Attribute), Subject (Kind Object).

ContextKind: SubjectKind (Attribute), ObjectKind (Value). Context (Statement) "signature" (dataflow inputs / outputs activation: domain / range).

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

Resource : Functional (Monad) OntResource wrapper.

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

(Statement, Occurrence, Attribute, Value);

(Entity, Statement, Occurrence, Attribute);  
(Role, Entity, Statement, Occurrence);  
(Class, Role, Entity, Statement);  
(Flow, Class, Role, Entity);  
(Behavior, Flow, Class, Role);

Class intension / extension (Context / Value).

Model levels (layers reification). Ontology hierarchy: from upper ontology to (abstract) user interface components.

Data order: Resource Kind hierarchies.

Schema order: Role Class hierarchies.

Interaction order: Statement Context hierarchies.

Form / Flow Template Matching: Alignment / Encoding. Populate Template with Message. Map.

Augmentation Mappings: Flows (Wrapper API). Exchange. Reactive IO Resource Oriented.

Classes: Layers monads. Class hierarchy. Inputs resolves from wrapper containers to next layer occurrences (map forward), occurrences contexts collects matching result graph (reduce backwards). Map / Reduce: Graph key / value / properties encoding.

ID<ID> : Reified matching URIs.

Message (encoded reified verb?)

Transform<ID> : Augmentation Range

Mapping<Transform>

Template<Mapping> : Augmentation Domain

Augmentation<Template> : Domain / range signature

Resource<Augmentation>

Role<Resource> : CSPO Role

Statement<Role> : CSPO Quad

Model<Statement> : Set of Statements Role Resource Occurrences (Kind?)

Messages: Augmentation (performed transform, Flow), Template, Mapping (possible transform, Form), Transform. Dialog.

Model Reactive I/O:

Model forward (map inputs): aggregate inputs into reified layers contexts instances (Model Meta Resources reification).

Augmentation: populate / perform Flows. Aggregate, Align, Activate (over mapped inputs). Mapping Template Transform algorithms / services encoding in Statement plus Meta Resources.

Model backwards (reduce outputs): collect occurrences graph (matching signatures contexts from Model layer to IDs).

Encoding: nested shapes of recursive cuads (till primitives). Patterns / expressions: wildcards, variables, placeholders:

[[123, 456, \_b, $a][\_b][\*][$a]]

Resource class / component kind:

Members. Relations: Graph quad layers bindings (DOM). Previous, next, parent, child (order: class hier relations), Resource (instance), Role (metaclass), Statement (occurrence), Kind (class). Resource Monads. Eval rels axis: instance. Functors: ID Monads rels traversal.

(Context, Occurrence, Attribute, Value);

Events API. DOM. Monads. Functors (domain / range). Relations / traversal. Events. Encoded in Meta Model (Message Monad).

Functional Object Model:

Events (Message I/O) conditions matching (resolution / instantiation / bindings).

Objects: componed of aggregation of monad resources of a reified ID.

Flows: (Resource (Role (Statement (Kind))));

Forms: (Template (Mapping (Transform (Augmentation))));

Object: (Value (Attribute (Occurrence (Context))));

Members, relations, endpoints (API).

Message I/O: Dispatch according signatures bindings. Augmentation events (Functors) Kind streams.

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

Context: Resource / Message Monad Events: Augmentations. Mapping: Endpoint. Events: Implement Message / Resource / URIs Protocols.

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

Order: Model Resources react to events according Message matching event “signature” (Kinds).

Message (Subject : Data level)

Template (Context / domain : Session level)

Augmentation (Occurrence, declarative / service Resources: functors. Interaction level)

Transform (Role / range: Kind transform matches. Session level). Resulting Message Attribute / Value roles populated.

Meta Model encodes:

Meta Resource class / instance patterns.

Participation: Subject in Occurrence.

Role: Participation for Subject.

Kind / Context hierarchies.

Mappings: Facets (Models / Contexts declarations) by Meta Resource statements in Meta Model. Mappings renders Model(s) contents statements (layers) by Context Augmentations.

Augmentations defined as declarative Mappings in Meta Model encoding Context (layer) inputs matching signatures and augments current / previous layer emmiting mapping transforms. Context : Functor. Participation wraps Context / Resource.

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.

**Layer Levels**

Models / Meta Model Levels: reify models Contexts hierarchies into IDs layer. For each model layer into different hierarchies reify an ID layer quad statement corresponding to those layer Context and perform corresponding layer(s) aggregations. Inferred “upper” ontologies. Mappings population / resolution.

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.

Examples: Source, Session, Interaction declarative application protocol use case upper ontology levels (Action… Gesture, etc).

**Encoding**

Model Encoding: Property graph. Properties (prefix codes, key / value, reification). Sets, groups, categories. Functors applications: Transforms as graph navigation / browse. Template Message parsing (grammar, verbs, state flow). Contextual Quad Context ID: ID according occurrence in Statement context (normalized forms). Occurrence Context IDs indices / mappings.

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.

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

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.

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.

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). ToDo.

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.

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: 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: Saga Activation. Interaction Model (Meta Model). Aggregated (Interaction) Meta Model interactions (performed / inferred / possible) emitted as Model event Messages (Saga pattern). Mappings. Event sourcing.

Resolve Message matching Resource from behavior layers / matching kinds from Model / data layers.

(Kind, SuperKind, Attribute, Value);

(Occurrence, Kind, SuperKind, Attribute);

(Context, Occurrence, Kind, SuperKind); (attributes / links bindings).

(Resource, Context, Occurrence, Kind); State Resource Kind in occurrence context (context / role bindings).

(Statement, Resource, Context, Occurrence); State Resource URIs occurrences / Resource class IDs (classification bindings).

(Interaction, Statement, Resource, Context);

(Action, Interaction, Statement, Resource);

Example: a message composed of a kinds CSPO matches statements “instances” of those specifications (statements whose CSPO have matching kinds). A message with three CSP kinds and a (potentially unknown) object URI retrieves matching resources having that object value into corresponding property kinds. An statement of plain (potentially unknown) URIs instantiates / updates and augments new / known resources added to models and returns an augmentation transform result.

Interaction Model: Context of Messages model for a given interactions session / dialog state. Message invocation requests: Statement(s) building Resource invocation graph with layers matching Message patterns. Layers graph invocation patterns matching from higher to lower layers resources fulfilling higher layers templates. Variables, wildcards, placeholders.

Explain messages (resource resolution). Grammar. Match model Resource(s). Compound nested CSPO statement contexts defines result behaviors. Message CSPO contexts may define create, retrieve, update or delete operations (passing 'null' for example for resource / statement to be deleted). Forms / Flows.

Explain transforms (message application). Transform: Resource stream result of Message application over resolved Resource(s)). Input statements: Message(s) / Resource(s) (from input message or to be populated or populated in dialog) and "goal" Message / Resource aggregating a model from Resource MetaGraph with Message / Resource bindings.

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).

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 possible into distributed peers which exposes protocols / APIs for knowledge driven hypermedia applications.

**11.2.1.: Aggregation**

Aggregation: infer roles in contexts: regression (Person class in Employment interaction : Developer role).

**11.2.2.: Alignment**

Alignment (infer attributes / relations): clustering (from multiple occurrences of same entity in diverse data sources).

**11.2.3.: Activation**

Aggregation: infer roles in contexts: regression (Person class in Employment interaction : Developer role).

**11.3.: Augmentation: Events Mappings Realizations**

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).

Interactions declarations: signature definitions (Template / Transform contexts). Interaction instances: addressable exchanges (Augmentations, Message, Model context / Mapping bindings / matchings / performances). Meta Model / Levels event driven Model Augmentation.

Interactions declarations: signature definitions (Template / Transform contexts). Interaction instances: Exchanges (Augmentations, Message, Model context / Mapping bindings / matchings / performance). Contexts / Exchanges: Meta Model / Levels event driven source Augmentation events declarations (populating Facets / Layers / Levels).

## Models / Meta Model IDs

IDs:

URI(s);

OntResource; Merged URI(s) wrapper.

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

Resource (OntResource Context Roles hierarchies Monad wrapper);

Statement : Resource quad, Resource.

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))));

ToDo.

**Implementation:**

Definitions: Quads, contexts, Kinds, Grammar, etc.

Implementation: Runtime. Architecture. Components. Patterns. Models. Messages. Augmentation. Events dispatch. Message aggregation / instantiation / resolution / application. Backends. Services.

Resources inputs / outputs: Augmentation, Protocol, Browser. Message addressing / resolution / application.

Component: Models (data). Source: augmented input statements. APIs (Model).

Component: Messages (contexts). Source: augmented models templates (Grammar). APIs (Model).

Component: Transforms (interactions): Source: input statements case matching Message inputs. Returns / materialize results. APIs (Model).

Core Model API: Augmented (Aligned, Activated, Aggregated inputs matching model context messages) IO. Resource MetaGraph. Dimensional model. Grammar. Model repository. Backend. API.

Core API: Model, URI, Resource, Statement, Kind.

Message: Context Model API. Input statements: Model Grammar. Augmented IO by interaction transforms of applied matching Message with model statements inputs. Context of core models instances. API.

Transform: Interaction Model API. Input statements: Transform request invocation specification. Functional application of Message(s) over Resource(s): Transform (streams). Augmented IO: Requested Transform which applied augments resulting responses (dialog arguments resolutions). Context of context model instances. Reactive / streams API.

Message Transform (interaction result): matches request context specification built upon Resources / Messages (TransformBuilder). Resolve state / dialog session graph. Returns observable stream. Dataflow (chaining). Operations (over streams).

Transform request invocation specifications: means to interact with underlying contexts models (CRUD, domains behavior). Transforms result from applicating Message(s) over Resource(s). Sending a Message Resource to a given interaction context initiates a “dialog” in which to “populate” target Resource(s) and Resource arguments. Each dialog “step” renders resources / layers streams of requested arguments (server “queries” clients) or resources / layers streams of response augmented Resource(s).

Message IO encoding components kinds:

Data: Assertion (statement / entity).

Schema: Type (kind / class).

Behavior: Interaction (flows / behaviors).

Navigation semantics: referring contexts.

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.

Functors: Meta Model declarations / Context classes / instance declarative implementations. Aggregation type: invocation over each CSPO / Context roles.

Functor applied to context: Aggregation.

Functor applied to subject: Alignment.

Functor applied to predicate: Activation.

Functor applied to object: members traversal.

Reactive Context Kind (matching signatures) dataflow.

Message - Model - Template (context) - Augmentation (interaction) - Transform (data) - Model - Message.

Implementation API: Node / Container. Services (URIs Context Kind signatures resolution).

Core Services: Activation Augmentation (Naming).

Core Services: Alignment Augmentation (Index).

Core Services: Aggregation Augmentation (Registry).

Core Services: RDF / OWL Backend (endpoint, reasoning, persistence).

Core Services: DIDs Persistence (sync Node state: events sourcing).

Core Services: Protocol (I/O). Node, Session, Intetaction levels. Base Connector Augmentation API. Event driven URIs dialog / prompts protocol adapters.

Explain Context layers Aggregation Augmentation. Example: Role(s) for each CSPO. Entity in Statements. Meta Model. Meta Resources.

Explain Context layers Alignment Augmentation. Meta Model. Meta Resources.

Explain Context layers Activation Augmentation. Meta Model. Meta Resources.

Explain Augmentation. Context : Functor<Template, Transform>; Template, Transform : Context Kind (Levels: Data, Session, Interaction / Functor instance / execution contexts?).

Functors: Meta Model declarations / Context classes / instance declarative implementations. Aggregation type: invocation over each CSPO / Context roles.

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).

**Ontology Matching**

Ontology matching. Dataflow: sort statements. 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. Meta class, class, instance, occurrence.

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.

Ontology matching (Data, Schema, Behavior alignments).

Message (Form / Flow) / Resource: Meta Model parent classes? Specification / Protocol. Signatures: Mappings Context Kinds. Possible Flows (Form), actual Augmentation (Flow).

Basic hypermedia browse / CRUD (HTTP verbs) bound Message functors compatible for all Resources (REST).

Resolve Message / dialog (CRUD) semantics via MetaGraph driven transforms (data / schema / behavior augmentation: dialogs).

Events: Monads (IDs Contexts hierarchy instances), Functors (layers classes instances reifying model classes / domain instances from facets / levels). Augmentation: materialized Transform. Flow: Mapping possible Transforms. Browse / Apply (generic flows?). Dimensional.

Classes: Layers monads Contexts cllass hierarchy. Inputs resolves from wrapper containers to next layer occurrences (map forward), occurrences contexts collects matching result graph (reduce backwards). Map / Reduce: Graph key / value / properties encoding.

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).

Component. Services. Protocols. Archetype Reactive Functional (Monads) Component APIs. Reactive: Connector / Client Endpoints: Consumer / Producer / Processor (Service / Model) inputs / outputs handlers (formats / protocols parsing / matching / alignment into IDs / Contexts. Reactive Augmentations: fire possible dataflows).

Environment: Models events abstraction (subscribe / augment / publish) Connector / Model / Client Augmentations IO.

Encoding: XML / XSL / Template Scripts (functional runat: peer dialogs / reactive callbacks). Mappings declarations / encodings (primitives, wildcards, variables, placeholders templates: actual / result of, possible).

Streams: URIs, Resource, Statement, CSPO Roles, Kinds. Dataflow: index / signatures dispatch, reactive.

Formalization: Functional / Object APIs. Reference / Data model. Sets, categories, models. SortedSet (hierarchical structures).

Kinds, Signatures. Contents. Contextual metadata. Lattices. Roles / Sets (bitstring quads). Definitions (elements). SortedSet (hierarchical structures). Key / Value graph encoding. Map Reduce. Flows: Mapping declarations / assertions (possible flows).

Operations. Rules. Categories. Groups.

Semantic resolution: Query Resource(s) satisfying “criteria” (i.e.: Object(s) for predicate / Augmentation Mappings Forms / Flows) IDs by IDs resolution pattern: (Message applicable signatures : resolution result: Transform).

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

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.

Semiotic:

Subjects: attributes / values. Occurrences: contexts / roles.

(Context, Occurrence, Attribute, Value);

(Context, Sign, Concept, Object);

Metaclass, class, instance, occurrence.

Assert order / hierarchies / relations in dimensional axes. Containment (sets).

Messages: Service Context URIs: Signature for face recognition (image URI / resource : domain, detection / search results endpoint / placeholder : range). Others services: ML Classification, Clustering, Regression, Services Index, Naming, Registry. Presets "inferred" models and augmentation services (populated / online learning).

Augmented Semantic Content Types (img/xml;facesCoords).

Upper Ontologies. Load. Grammar level services (schema browse, possible flows query / browse). Message: wildcards, variables, placeholders.

Dimensional:

Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements layers).

Value -> distance(prev, next); ordering;

Assert knowledge: 1h -> 60min;

Assert: dom-lun-mar-mie-jue-vie-sab;

Assert: 1mt -> 100cm; etc.

Comparison / order: Alignments (prev, curr, next asserted knowledge). Next hour, location, city, country, next distance at next time at current speed. Event sourcing / tracking: married -> marriage occurred.

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). ToDo. SortedSet hierarchies.

Sort: cause / effect, temporal, etc. Messages align, functional map, fold, etc. Primitives. Encode layered statements ordering. Complement / supplement concepts definitions.

* (Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.
* (Measure, Value, Previous, Distance);
* (Unit, Measure, Value, Previous);
* (Dimension, Unit, Measure, Value);
* (Concept, Dimension, Unit, Measure);
* (Resource, Concept, Dimension, Unit);
* (Statement, Resource, Concept, Dimension);

Populate / align / annotate models with dimensional data. Model input: statements (model resources). Model specification: augment, sort statements. Model specification: specialization of base model layers. Resolve resolution statements order.

Dimensional input set model specificatíon (from Statement layer, ordered SPOs: order criteria, comparisons. Kinds / class / occurrence / instance order criteria?).

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 matching (table, pk, col, val example). Helper upper models for models linking / alignment. Shapes. ISO.

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

Ontology matching: Dataflow: sort statements. Units. Equivalences. Distances / events (order). Services (Augmentation / Context Functors Meta Model mappings / transforms). OntResource; Merged URI(s) wrapper. OntResource hierarchy: layers statement contexts. Facets DOM, Actor / Role. Resource (OntResource Context Roles hierarchies Monad wrapper); Statement : Resource Role quad, Resource.

Ontology matching: Events declarative definition. State change of value in axis in measure of context. Dimensional Model.

Ontology matching (table, pk, col, val example). Helper upper models for models linking / alignment. Event sourcing (“offline” sync). Graph linking / alignment / sinchronization by entailments from event sourcing over inferred state. Reconciliation.

Messaging metamodel:

(Message, Resource, LHS, RHS);

(Interaction, Message, Resource, LHS);

(Role, Interaction, Message, Resource);

(Context, Role, Interaction, Message);

(Dataflow, Context, Role, Interaction);

Encoding:

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. SortedSet hierarchies (3 digit octal set membership values).

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

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

Quad encoding: Context relative IDs.

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). ToDo.

Dataflow: Order, Forms, Flows (Signatures, Mappings, hierarchies).

State order (in context class hierarchies axes), comparison relations, iterations, flow, events, causal relations, units, enums, equivalence, etc.

Data order: Resource Kind hierarchies.

Schema order: Role Class hierarchies.

Interaction order: Statement Context hierarchies.

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

Meta Resource / Models / Messages: IDs / Encoding / Addressing formats. Ontology matching and Template / Augmentation / Transform enrichment (alignments), transforms (functors), materialization (model updates) via Mappings (events) and Meta Resource / Model Encoded Resource declarations (enrich / align, transform, updates algorithms: Encodings).

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.

Backend:

Messages: Saga Activation. Interaction Model (Meta Model). Aggregated (Interaction) Meta Model interactions (performed / inferred / possible) emitted as Model event Messages (Saga pattern). Mappings.

Messages: Events IO / Persistence: Saga Activation / Passivation populating Node local Quad store / persisting peers via DIDs (inference enabled distributed consistency) semantic (resolvable / discoverable) identifiers.

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.

Goals: Low Level P2P Services. Intermediate API (Forms / Flows) for OGM. Semantic Browser.

Event sourcing / tracking: married -> marriage occurred. Dataflow: Messages hierarchy. Aggregate contexts from coarse to fine grained  transforms (raiseSal -> setAttr, single – marryWith). Listen mariage event (update salary to familySalary).

Message dispatch, input statements resolve to applicable messages from switch from behavior to data layer invoking async microservice. Message case matching may involve entering and leaving data, schema and behavior paths if aggregated contexts matches more than one message. Visitor.

Message: functor (monadic transform) : Resource<T> -> R, T, R : URIs (hierarchies, models, semantic content types). Available verbs / flows / navigation (browse models, state of application returned from materialized models). Parameterized functions (partial applications) into Messages metamodel resources. Contexts (dataflow). Execution graph.

Subscriptions declarations / definitions. Applied on streams activations (transforms, executions resource parameterized partial contexts).

URIs Services: API for plugging whatever connector may be implemented for behaving in a reactive message oriented fashion (back ends).

Resource: Abstracts (wraps) URIs Services in a functional API (Resource streams). DOM, Actor / Context / Role (Meta Resources).

Super Kind / sub Kind hierarchy relationship is given by a set of Kind Attributes being super set / sub set of each other. (Sub, Super, Attr, Value).

SubjectKind (meta Resource): For a given URI occurring as Subject (Occurrence) across a set of Statements (Contexts), its aggregated Predicates (Attributes) defines its "Kind" and its Attribute values determines the given Kind instance "members" values.

ObjectKind (meta Resource):  for a given URI occurring as Object (Value) over a set of Statements, Subject (Kind Attribute), Predicate (Kind Value).

PredicateKind (meta Resource): for a given URI occurring as Predicate over a set of Statements, Object (Kind Attribute), Subject (Kind Object).

ContextKind: SubjectKind (Attribute), ObjectKind (Value). Context (Statement) "signature" (dimensional order / dataflow inputs / outputs activation: domain / range).

Extended content types activations on domain / range (verbs, augmentations). Example: image, face, crop.

Encoding: Kind hierarchies / Grammars

(CK, SK, PK, OK);

Semiotic / Dimensional Alignment, Aggregation (known mappings)  : Class / ID Ontology Matching. Contextual IDs (infer occurrence contexts). Inference ID lookup of ID for desired satisfaction of given transforms / roles / operations.

Graph Execution Semantics: Dataflow by Context Kind domain (Subject Kind) / range (Object Kind) matching Forms / Flows. Ontology Matching. Upper ontologies. Primitives.

Encoding: Resource ID. Encoded Resource contents (signature / occurrence). Augmentation: Resource (SortedSet) set (Message) resolution from context over Template / Resource(s).

Encode IDs: Context Kind, upper (meta) Resources (levels / layers). Resource contents / contexts (identify by occurrences in roles in other contexts, Meta Resources, layers class, metaclass, instance). Compose IDs (hierarchical graph properties encoded string) from outer to inner resources (Context, Kind, Occurrence, Role, Resource). "Operable" IDs (ClassIDs / InstanceIDs: Meta Model reifications / occurrences).

Encode common upper Semiotic / Dimensional Model: Reference Model.

Encode Kind / Context hierarchies.

Encode Augmentation(s) as Resource descriptions.

Encode Model(s) as Respurce set. Meta Resources, layers Contexts, Kinds (reified).

Encode Graph Execution Semantics. Dataflow: Context Kind signatures. Iteration, conditional jumps.

Object occurrence of Predicate.

Sets. Quads. SortedSet.

Metaclass / Class / Instance.

Class / Instance ID pairs:

Subject / Context / Role : Attribute, Value. Metamodel. Encoding: each type as each (pair) kind. Pairs.

Semiotic encoding:

(Context, Sign, Concept, Object);

Value as Occurrence of Attribute in Attribute Occurrence Context. Meta Resource context roles).

Augmentation: basic operation. Resource Set Specification (SortedSet / Statement) matching Model which returns augmented Message response (Model I/O).

Encoding: recursive resource quads encoding hierarchy, order, class, instance, attributes. Operate inferences over (upper) patterns (bitstring / lattice). Meta Model, Facets, Levels. Specifications: Signatures, Forms, Flows (encode events / transforms provenance).

Message: Resource aggregation (occurrence, context, model) dataflow (Augmentation). Resolves Resource Set specification.

Dataflow: Message - Model - Template (functor) - Augmentation (interaction) - Transform - Message - Model

Order: Common super type / kind / role / occurrences. SortedSet.

Augmentation: common super type inference / alignment: Aggregation, Alignment, Activation. Verbs / Activation. Functors (context: messages, reified mappings: templates).

Message: specification / transform (input / output dialog domain / range). Context Kind.

Augmentation: Aggregation (Context template).

Augmentation: Alignment (Attribute, Value template).

Augmentation: Activation (Kind type inference, Class / ID resolution / alignment: semiotic / encoding templates).

**TOC:**

**Distributed Integration and Consistency for Knowledge Semantics**

[Introduction 1](#_gjdgxs)

[Metamodel 3](#_30j0zll)

[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)

Introduction

Distributed Integration and Consistency for Knowledge Semantics.

Property graphs allow properties (key/value pairs) to be associated with both nodes and links in directed graphs. This allows you to annotate links with information such as the start and stop times for when the link is valid, its provenance, a statement about its quality and so forth.

Property / labeled graphs. Dot notation / encodings. Nodes / arcs links / attributes: metaclass / class / instance / occurrence levels. Individual node / arc label or other node / arc in rel context (subjects in context occurrence attribute / value data model). Links / rels subjects (with labels).

Signatures: Context / Subject, Occurrence, Attribute, Value. Metamodel: metaclass, class, instance, occurrence upper ontology (primitive) property graphs categories / types.

Functional. Sets, groups, categories. ADTs. Metamodel. Augmentations / Functors. Layers, Aggregation, Alignment, Activation (type) categories / transforms (encodings).

ERP Ontology: from concepts to application gestures / purposes (high level APIs).

Description

Distributed Knowledge Base for Functional Syndicated Application Integration and Virtualization Framework. Plug existing backends (applications / datasources / services) via in an EAI / ESB fashion. Provide semantic augmentation of learned applications metadata (data / schema / behavior) via Augmentations (Aggregation, Alignment, Activation).

Distributed systems / micro services access to shared data. Shared data consistency / inference / MDM. Ontology matching. Integration features: rules and flows based services composition.

Distributed P2P (Blockchain) approach of data synchronization between peers for ease of deployment patterns election and datasources integration (APIs, microservices, etc.).

Augmentations:

Aggregation: infer roles in contexts (see Metamodel Layers). Regression: Person class in Employment interaction referring context, Developer role.

Alignment: infer links / attributes. Clustering: from multiple occurrences of same entity in diverse data sources, complete missing / new information.

Activation: type inference. Classification: determine metaclass / class / instance / occurrence subject roles for corresponding entity attributes and values.  
  
Hypermedia APIs for reactive integration of addressable resources representations. Content type driven semantic augmentation / annotations. Plugged backends synchronization. Data, information, knowledge levels of interaction contexts abstractions (from dimensional to plain facts).  
  
Domain Driven Development. Use cases "problem spaces" translation of aggregated domains services into underlying resources. Domains Use cases abstractions (problem "spaces" / ontologies) enabling domain translation / exchanges / integration. Declarative Application Design.

Applications:

Hypermedia Dataflow Activation (reactive / event driven knowledge based contents). Dataflow layers.

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

Connected application sources (backends: EAI / ESB) and declaratively stated application models.

# Metamodel

Layers are implemented as an RDF Quads hierarchy aggregating each one on top of another. The idea is that aggregating Data according some criteria one could enable us to infer the Schema that those Data belongs to and that aggregating Schema and Data one could enable us to infer the Behavior (operations) that correspond to the Data manipulation in that corresponding Behavior layer class / instance.

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.

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.

Link grammar: (employer, employee). Categories, monads. Reference Model.

Link Grammars. Types: links left / right types defined when a shape / slot match satisfaction occurrs (roles).

Parsing: extract propositions, knowledge assertions (in a domain ontology Behaviors). Assert propositions links, order, concepts relations (between domains). Link Grammar. ISO TMDM / TMRM.

Parsing: extract prescriptions, knowledge rules (in domain ontology concepts relations: causal, requirements, etc.). StratML.

Parsing: infer possible statements propositions / prescriptions productions. Link Grammar Disjuncts. Embeddings.

Message: Functor Declaration. (events / grammar: protocols).

Augmentation: Functor Instance.

Context hierarchies: categories of wrapped subject occurrences. Statement Context monad (category) for Subject occurrences.

Context category instance identity: Employee Kind, Work Behavior, etc. Context statement attribute / value aggregation. Key / value abstraction scoped in category type / instance.

Functors: Layers (Context categories) aggregation / transforms. Augmentations.

Augmentations: Categories Aggregation (types), Alignment (contexts), Activation (roles / matching in interactions).

Locators (keys): (metaclass, class, instance, occurrence);

Metaclass: Transform OntResource

Class: Message Augmentation

Instance: Statement Role

Occurrence: Flow Kind

Locators (keys): local / remote keys. Navigation. Paths. Legends. Signatures (streams).

Monadic (locators / signature streams based) category wrappers Functors:

M<T>::flatMap(functor(T) : U) : M<U>;

Dado rango y alcance, universo: U de una relación R, inferir dominio y codominio, campo: C.

From Object (O) extension /instances to Context (C) intension / class.

Aggregation (types) functor signatures: stream.

Alignment (contexts) functor signatures: stream.

Activation (roles / matching in interactions) functor signatures: stream.

Encoding: IDs. Embed metaclass, class, instance, occurrence metadata (context, role, attributes, values). Functional APIs. Wrappers / Transforms (augment: aggregate / classify, roles, properties "graph" rels). Polygon Vector Space Model. ANN embeddings / autoencoders.

Forms / Flows Dialogs / Contexts. Protocol. Resources, addressing, representations, navigation / traversal: properties "graph" rels (Wrappers / Transforms). Functional APIs.

Sets encoding: properties in axes (kinds). SortedSet (hierarchies). Metaclass, class, instance, occurrence properties in axes for CSPO IDs. Augmentations: property graph rels navigation / traversal. Dialog Forms / Flows "state" contexts.

Contexts Wrappers kinds Transforms / Traversals functors: Augmentations declaratively stated in upper Context layers (kind classification, kind roles, kind attributes / values).

Dialog Forms / Flows "state" Contexts browsing (upper Context SPO kinds: current context streams).

Augmentation navigation of Transforms / Traversals as a Context (streams / filters). Levels / reification.

Order. Iteration. Predicates (resource meta / domain / kinds). Streams filter, conditionals, jumps. Aggregation. Functional mapping / reduce, etc.

Resource API (layer roles):

Statement: Resource (CSPO: Statement) Property graph URL wrapper. URL occurrences aggregate. Functional occurrences properties (roles / streams) for Statement wrapped URL APi:

Context: Metaclass URL occurrences.

Subject: Class URL occurrences.

Predicate: Instance / Occurrence URL occurrences.

Object: Occurrence / Instance URL occurrences.

Statement wrapped URL occurrences functional roles Kinds API: Map<Role, Kind> (reified in Metamodel) in occurring Kind query / stream context selectors: ID (URL), layer type, layer role, layer kind occurrences context selectors.

CSPO Kinds (streams functors) declaratively stated in aggregated layers CSPO occurrences (kind classification, kind roles, kind attributes / values).

Augmentation navigation of kinds Transforms / Traversals as streams / roles (filters). Levels / reification: kinds from Statement / roles layers (reification / levels axes).

Query / stream context selectors: ID (URL), layer context type, layer context role, layers kinds occurrences. Transforms (functor kinds: augment / browse query context according kind specification with corresponding statements).

Levels: Grammars (kinds functors dataflow signatures). Productions: Augmentations (parsed / produced in navigation contexts). Dataflow order (sets / hierarchies).

Monad: context type. Metaclass.

Monad: wrapped type. Class.

Monad: wrapped value. Instance.

Monad: wrappers hierarchy context type instance. Occurrence.

Unit. Type constructor (hierarchy context / member types / values factories). flatMap / map / flatMapN.

Monad: context type case classes hierarchy? Factory methods, hierarchy types / members values / signatures (wrapped context / values) cases (predicates):

Optional: None | T [case context | case context | case value signature]

Result: Error | OK [case context | case context | case value signature]

Writer: Value, List<S> trace.

Dataflow:

Input statements: Augment. Aggregate, build layers contexts representation quads. Parse:

Monadic parser combinator: aggregated metamodel nested layers contexts corresponding wrapper / wrapper hierarchy types DOM / AST (from contexts quads aggregation nesting hierarchy levels).

Augment: signatures dataflow. Apply input statements: new inputs / kind new attributes (person, employee -> position, salary). Parse.

Protocol: Forms / Flows augmented / parsed representation / metamodel I/O. HAL / HATEOAS endpoint encoding for navigation, transforms and inputs augmentation.

Protocol: Forms / Flows Dialog / Prompts resolution. Context roles, wrapper kinds navigation / transforms declaratively stated in encoded representations.

Actor / Role. Dynamic Object Model (OGM / Kinds). Golden Braid: Metaclass, class, instance, occurrence relation relative to layer context levels.

Reference Model:

Type : Value;

(Value, Type);

(Type, Type);

(Value, Value);

Aggregation: cons cells tree traversal. Layers contexts representation. Dataflow augments updates / append corresponding tree cells: (first: (rest: nil). (C (S (P (O))));

Aggregate / Augment inputs / transforms: Parsed model streams. Reactive data structures.

Encoding: addressing, semantic graph networks. URN overlay semantic addresses encoded mappings.

Functor: mapping between categories A / B using a function. Map, alcance, rango, dominio, imagen (infer, aggregate). Connections: same number of items (inyective, biyective). Function: de wrapper en wrapper (morphism new image). Message (functor) / Augmentation (transforms) Metamodel reifications.

Productions:

Inputs / Outputs: aggregate Augmentation / Message from input semiotic alignments. Protocol.

Inputs / Outputs: HATEOAS protocol browse / transform aggregated Context (model) Messages / Augmentations.

Inputs / Outputs: Layers hierarchy Augmentation / Message population / transforms. Meta Resources (Message / Augmentation reified context recursive schema / transforms declarations) I/O.

Monads: Quads Contexts wrapper of Occurrence / Subject aggregations. Root type: Resource. Kind (model / semiotic / domain meta resources) functors filter / traversal (streams: flatMap Resource set specifications).

Wrapped resources holds references to its wrapping occurrence contexts (Resource root type interface, contexts hierarchy / reification levels: upper layer instances reify layer occurrences in contexts). Context siblings, parent / child hierarchies kinds / attribute / values browse / filter / streams (axes: type activation, roles in context, attributes / values augmentations).

Wrapping unit / bind: layers traversal until container wrapped category type is met. Example: Behavior(s) of an Entity. Augmentations (types, roles, attributes reified model / domains aggregations). Example: Model, SK, P; Domain, someSubjKind, somePredicate.

Form / Flow: state navigation / browse. Dialogs / Prompts (contexts). Messages I/O: Augmentation signatures (kinds) streams.

Monads: Messages. Augmentations according signatures. Broadcast. Key / Value, Event sourcing routes. ServiceMix / JXTA: local Peer OSGi bundle. Camel ActiveMQ routes through local peer (jxta://localhost).

Messages: Augmentation according signatures. Broadcast. Key / Value, Event sourcing routes (levels). Form / Flow Dialogs / Prompts Messages for signatures (kinds) of unknown Resources resolution (layers wrapping traversal, CSPOs layers resolution for CSPO signatures / attributes / values).

Layers hierarchy:

(Context, Object, Concept / Sign, Value);

Semiotic roles. Kinds, grammars, levels. Contexts aggregations.

Resource: Plain inputs.

(Resource, Resource, Resource, Resource);

Role / Kind: Aggregate input statements CSPO roles (S) and their corresponding Attributes / Values.

(Role, Resource, Resource / Attribute, Resource / Value);

Statement: Aggregate Resource occurrences in CSPO Role(s) for Attributes (Kinds).

(Statement, Role, Resource, Resource / Attribute);

Entity: Semiotic Sign (Concept) Value (OntResource). Aggregate equivalent (ontology matching Resource extension for Entity intension.

(Entity, Statement, Role, Resource);

Template: Grammar kind. Semiotic Concept / Sign. Layer instance "template" kind (transform). Object occurrence. Aggregate Entities with similar kinds (properties). Occurrence Role / Kind.

(Template, Entity, Statement, Role);

Mapping (Instance) : Semiotic Object. Layer category type instance. Statements which are results of Behavior Flow (Statement: extension, Mapping: intension). Aggregate set of statements regarding the same concept (Mapping) in different occurrences.

(Mapping, Template, Entity, Statement);

Flow: Event declaration (I/O). Entities which are results of Behavior Flow (Entity: extension, Flow: intension. Mappings objects in Template Kind roles). Semiotic context.

(Flow / Augmentation, Mapping, Template, Entity);

Behavior: Message Specification (I/O).

(Behavior / Message, Flow, Mapping, Template);

(Measure, Behavior, Flow, Mapping);

(Unit, Measure, Behavior, Flow);

(Dimension / Axis, Unit, Measure, Behavior);

(Value, Dimension, Unit, Measure);

(Concept, Value, Dimension, Unit);

(Object, Concept, Value, Dimension);

(Context, Object, Concept, Value);

Inputs feedback. Semiotic encoding (feedback / grammars / levels).

Meta Resources (Message / Augmentation reified schema / transforms aggregation / recursive declarations).

Aggregation: materialize grammar statements of contexts recursively. Activation (types). Alignment (augmentations).

(Aggregation, Message, Augmentation, Mapping);

(Alignment, Aggregation, Message, Augmentation);

(Activation, Alignment, Aggregation, Message);

Encoding. Model layers:

URIResource context: CSPO form. RESTful / HAL monad: HTTP category functors.

Inputs / Outputs. Semiotic encoding (feedback / grammars / aggregated levels):

(Context, Object, Concept, Sign / Value); Semiotic roles (object as context / value, value as object, etc.) Labeled property graph / lattice. Inference rules (transforms) over roles. Kinds, grammars, levels.

OntResource context: (Resource, Occurrence, Attribute, Value) form. Aligned (matched) URIResource(s).

Resource context: OntResource (aligned / matched URIResource) occurrences in reified Role in Statement.

(URIResource, URIResource, URIResource, URIResource);

(OntResource, URIResource, URIResource, URIResource);

Aggregated URIResource OntResource attributes / values (recursion to attributes / values OntResource).

(Transform, OntResource, URIResource, URIResource);

Augmentation (production) Resource. Semiotic Sign / Value.

(Template, Transform, OntResource, URIResource);

Augmentation (production) grammar kind. Semiotic Concept.

(Mapping, Template, Transform, OntResource);

Semiotic Object.

(Augmentation, Mapping, Template, Transform);

Production (grammars). Semiotic Context.

(Message, Augmentation, Mapping, Template);

Grammar rules. Functors.

(Context, Message, Augmentation, Mapping);

Model.

(Resource, Context, Message, Augmentation);

(Role, Resource, Context, Message);

Reified CSPO / Resource, Occurrence, Attribute, Value Resource role types in Resource occurrence / context.

(Statement, Role, Resource, Context);

(Entity, Statement, Role, Resource);

Aggregated "subject" occurrences of Resource in Role in Statement(s).

(Class, Entity, Statement, Role);

Aggregated Entity Role occurrences type (attributes).

(Kind, Class, Entity, Statement);

Aggregated kinds / roles ("interfaces") of Class occurrences.

(Flow, Kind, Class, Entity);

Action "instance". Entity of Class performs role (Kind) of Behavior Flow.

(Behavior, Flow, Class, Kind);

Action "class". Statements: propositions, prescriptions, rules, productions. DCI / Link Grammar. Context satisfaction (rules).

(Measure, Behavior, Flow, Class);

(Unit, Measure, Behavior, Flow);

(Dimension, Unit, Measure, Behavior);

(Value, Dimension, Unit, Measure);

(Concept, Value, Dimension, Unit);

(Object, Concept, Value, Dimension);

(Context, Object, Concept, Value);

Inputs feedback. Semiotic encoding (feedback / grammars / levels).

Statements: propositional, rules, reasoning. Reification. Value: OntResource (instance); Concept: class, Object: metaclass; Context: occurrence.

Order. Order: Grammars kinds sets / hierarchy (signatures). Common super type / kind / role / occurrences. SortedSet.

Flat map functor: Kind (non terminal) instance to be extracted from (Augmentation) context (matching rules) according Message specification inside corresponding wrappers until context category. Grammar (rules signatures, messages).

## Aggregation

Super Kind / sub Kind hierarchy relationship is given by a set of Kind Attributes being super set / sub set of each other. SortedSet (order, iterations).

SubjectKind (meta Resource): For a given URI occurring as Subject (Occurrence) across a set of Statements (Contexts), its aggregated Predicates (Attributes) defines its "Kind" and its Attribute values determines the given Kind instance "members" values.

ObjectKind (meta Resource):  for a given URI occurring as Object (Value) over a set of Statements, Subject (Kind Attribute), Predicate (Kind Value).

PredicateKind (meta Resource): for a given URI occurring as Predicate over a set of Statements, Object (Kind Attribute), Subject (Kind Object).

ContextKind: SubjectKind (Attribute), ObjectKind (Value). Context (Statement) "signature" (dataflow inputs / outputs activation: domain / range).

Extended content types activations on domain / range (verbs, augmentations). Example: image, face, crop.

Meta Model: Encode / reify Model(s) w./ Meta Resources and Model Context(s) hierarchies. Meta Model: Encode order, iteration, conditional flow. Dataflow.

Encoding: Kind hierarchies / Grammars

(CK, SK, PK, OK);

Semiotic / Dimensional Alignment, Aggregation (known mappings)  : Class / ID Ontology Matching. Contextual IDs (infer occurrence contexts). Inference ID lookup of ID for desired satisfaction of given transforms / roles / operations.

Graph Execution Semantics: Dataflow by Context Kind domain (Subject Kind) / range (Object Kind) matching Forms / Flows. Ontology Matching. Upper ontologies. Primitives.

## Augmentation

Functor application.

Resource Monad:

Encode protocol functors. Endpoint address activation behavior facades. Graph state / rels traversal: Monad encodes entire state location flows to current CSPO URL IDs state (traceability in interaction context rels). Abstract Form / Flow attrs / rels.

Functors resolution on API addresses URLs: resource monads rels / attrs activation.

Behavior layer renders domains possible aggregated augmentations / messages of model functors composition. Rendered in domain levels as concrete contexts operations: named context operations over abstract model functors behaviors.

Augmentations:

Aggregation: infer roles in contexts (see Metamodel Layers). Regression: Person class in Employment interaction referring context, Developer role.

Alignment: infer links / attributes. Clustering: from multiple occurrences of same entity in diverse data sources, complete missing / new information.

Activation: type inference. Classification: determine metaclass / class / instance / occurrence subject roles for corresponding entity attributes and values.

## Functors

Functor: mapping between categories A / B using a function. Map, alcance, rango, dominio, imagen (infer, aggregate). Connections: same number of items (inyective, biyective). Function: de wrapper en wrapper (morphism new image). Message (functor) / Augmentation (transforms) Metamodel reifications.

Flat map functor: Kind (non terminal) instance to be extracted from (Augmentation) context (matching rules) according Message specification inside corresponding wrappers until context category. Grammar (rules signatures, messages).

Transform (Resource).

Mapping (Occurrence).

Template (Grammar Kind).

Augmentation (Production).

Message (Rule / Functor).

Core rules: Augmentations. Reified CSPO roles (kinds, etc. meta resources) terminals / non terminals.

Domain rules: instance CSPO roles (grammar) terminals / non terminals.

## Messages

Functor declaration. Resource set specification.

Message: Resource aggregation (occurrence, context, model) dataflow (Augmentation). Resolves Resource Set specification.

Order: Common super type / kind / role / occurrences. SortedSet.

Message: specification / transform (input / output dialog domain / range). Context Kind.

Signature based bus dispatch / application.

Signatures

CSPO Context Kind (Statement Subject Kind + Object Kind). Context Dataflow domain / range (Context as reactive streams producer / consumer).

Routes / Dataflow

Routes: Dataflow pub / sub bindings between matching signatures.

Core Model and Domain driven Message flow layout (Mappings).

Event Bus / Messages / Addressing

Dispatch Event into Dataflow Route.

Messages:

Monadic Functional Statement (Resource) wrapper.

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

Augmentation result: Message / interaction layer matching / populated Transform Statement. Template, Mapping, Transform Augmentation Meta Resources.

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.

Persistence:

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.

## Dataflow / Activation

Levels: Grammars (kinds functors dataflow signatures). Productions: Augmentations (parsed / produced in navigation contexts). Dataflow order (sets / hierarchies).

Event I/O (Quad Statements). Augmentations application. Dispatch: Signatures (Kinds). Apply transform (materialize event). Emit events (endpoints / filters / criteria streams signatures). Update graph listening events according its contents dataflow.

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.

Dataflow:

Events: Message (functor) declarations matching Augmentation Mappings by signatures.

Basic hypermedia browse (CRUD / transforms). HTTP verbs bound Message functors compatible for Resources (REST). Domain aggregated functors application (signatures).

Encoding: XML / XSL / Template Scripts (functional runat: peer dialogs / reactive callbacks). Mappings declarations / encodings (primitives, wildcards, variables, placeholders templates: actual / result of, possible).

Formalization: Functional / Object APIs. Reference / Data model. Sets, categories, models. SortedSet (hierarchical structures).

Semantic resolution:

Query Resource(s) satisfying “criteria” (i.e.: Object(s) for predicate / Augmentation Mappings Forms / Flows) IDs by IDs resolution pattern: (Message applicable signatures : resolution result: Transform).

Query Resources by role in context.

Query Resources by attributes / values.

Query Resources by identity / type.

Streams. Signature filters:

Streams: URIs, Resource, Statement, CSPO Roles, Kinds. Dataflow: index / order signatures dispatch, reactive (signatures).

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

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).

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). ToDo.

Messages: Service Context URIs: Signature for face recognition (image URI / resource : domain, detection / search results endpoint / placeholder : range). Others services: ML Classification, Clustering, Regression, Services Index, Naming, Registry. Presets "inferred" models and augmentation services (populated / online learning).

Augmented Semantic Content Types (img/xml;facesCoords).

Upper Ontologies. Load. Grammar level services (schema browse, possible flows query / browse). Message: wildcards, variables, placeholders.

## Services

Base core services URIs (index, naming, registry). URI subclasses implementing / wrapping state for Resource monads offering protocols / addressing / content types / representations / backends / endpoints facades for services: DBs, (SOAP, REST, SPARQL, JCR), ML (predictions), inference, etc.

External services I/O modelled as Resources in layers contexts. Layers (session, dialog, etc.). Node, Peer, Client, Connector, etc. Reactive / Event Driven. REST HATEOAS (dataflow).

Applications:

Hypermedia Dataflow Activation (reactive / event driven knowledge based contents). Dataflow layers.

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

Connected application sources (backends: EAI / ESB) and declaratively stated application models.

Index, Naming, Registry.

MapReduce.

Blockchain DIDs.

MapReduce (Encoding).

Backend Connectors.

DCI / MVC: JDBC / OGM / ORM / JCA / Activation JAF / Process Flows (state) semantics.

Declarative hypermedia: REST / HAL / HATEOAS functional protocol.

CMS / Wiki (API / Protocol / DAV). Docs. Forms (Docs APIs)

# Encoding

Encoding:

Encoding: IDs. Embed metaclass, class, instance, occurrence metadata (context, role, attributes, values). Functional APIs. Wrappers / Transforms (augment: aggregate / classify, roles, properties "graph" rels). Polygon Vector Space Model. ANN embeddings / autoencoders.

Forms / Flows Dialogs / Contexts. Protocol. Resources, addressing, representations, navigation / traversal: properties "graph" rels (Wrappers / Transforms). Functional APIs.

Sets encoding: properties in axes (kinds). SortedSet (hierarchies). Metaclass, class, instance, occurrence properties in axes for CSPO IDs. Augmentations: property graph rels navigation / traversal. Dialog Forms / Flows "state" contexts.

Contexts Wrappers kinds Transforms / Traversals functors: Augmentations declaratively stated in upper Context layers (kind classification, kind roles, kind attributes / values).

Dialog Forms / Flows "state" Contexts browsing (upper Context SPO kinds: current context streams).

Augmentation navigation of Transforms / Traversals as a Context (streams / filters). Levels / reification.

Order. Iteration. Predicates (resource meta / domain / kinds). Streams filter, conditionals, jumps. Aggregation. Functional mapping / reduce, etc.

Encoding: metaclass, class, instance, occurrence (contextual / nested / orders / ops) CSPO IDs. CURIEs.

Encoding: Sets CSPO Contexts specification (sets quad encoding).

Encoding: Functor application. Predicate: functor behavior, domain: statement predicate, transform / range: statement object.

Encoding: Levels (OntResource context hierarchy) reification: Message as Predicate, etc. Resource Monad (context statement / signatures). Functor aggregation: levels (type, role, alignment).

Encoding: Grammars. OntResource hierarchy reification: rules (contexts) / non terminals (reified Predicates / Kinds). Aligned OntResource URLs: terminals. Augmentations: productions (functors).

Link Grammars. Types: links left / right types defined when a shape / slot match satisfaction occurrs (roles).

JAF. Index / Naming / Registry (HATEOAS Forms / Flows navigation / states): DCI / MVC Engine.

Dimensional:

Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements layers).

Value -> distance(prev, next); ordering;

Assert knowledge: 1h -> 60min;

Assert: dom-lun-mar-mie-jue-vie-sab;

Assert: 1mt -> 100cm; etc.

Comparison / order: Alignments (prev, curr, next asserted knowledge). Next hour, location, city, country, next distance at next time at current speed. Event sourcing / tracking: married -> marriage occurred.

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). ToDo. SortedSet hierarchies.

Sort: cause / effect, temporal, etc. Messages align, functional map, fold, etc. Primitives. Encode layered statements ordering. Complement / supplement concepts definitions.

(Value, Previous, Distance, Next);

Person, Single, Marriage, Married;

Man, Single, Marriage, Husband;

Woman, Single, Marriage, Wife.

Populate / align / annotate models with dimensional data. Model input: statements (model resources). Model specification: augment, sort statements. Model specification: specialization of base model layers. Resolve resolution statements order.

Dimensional input set model specificatíon (from Statement layer, ordered SPOs: order criteria, comparisons. Kinds / class / occurrence / instance order criteria?).

Encoding:

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. SortedSet hierarchies (3 digit octal set membership values).

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

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

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

Encoding / Dimensional example: role in context. X is Y for Z in W.

(Z (W (X (Y))));

(Man (Marriage (Man (Husband))));

(Hour (Minute (1 (60))));

Encoding, Dimensional, Meta Model. Units. Events. Order. Relations. Comparison. Input layers. Augmentation.

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

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

Model: Contexts class subjects instances occurrence role kinds attributes / values.

Quad encoding: Context relative IDs (polygon). 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). ToDo.

Dataflow: Order, Forms, Flows (Signatures, Mappings, hierarchies).

State order (in context class hierarchies axes), comparison relations, iterations, flow, events, causal relations, units, enums, equivalence, etc.

Data order: Resource Kind hierarchies.

Schema order: Role Class hierarchies.

Interaction order: Statement Context hierarchies.

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

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.

Semiotic / Dimensional alignment / aggregation layers (lower resource alignment layers):

Semiotic / Dimensional Alignment, Aggregation (known mappings)  : Class / ID Ontology Matching. Contextual IDs (infer occurrence contexts). Inference ID lookup of ID for desired satisfaction of given transforms / roles / operations.

## RDF Quads

RDF Quads / Object Mapping (DOM / OGM)

As RDF Quads encodes four URI values (CSPO Statement) an Object - RDF Quad elemental mapping could be implemented regarding an RDF Quad Statement CSPO as follows:

(C: Context, S: Occurrence, P: Attribute, O: Value);

where Context (C) is the URI of an Object Class identifier, Occurrence (S) is the URI of an Object Class Instance identifier and, aggregating same Class / Instance pairs, Attribute (P) and Value (O) are, respectively, Class Instance member (name, domain / range) and values for the aggregated (S) Object of Class (C).

Contexts. Occurrences, Attributes, Values: Roles of Meta Resource(s) in contexts.

Subject in Statement has Predicate and Object Attribute / Value (roles).

Predicate in Statement has Subject and Object Attribute / Value (roles).

Object in Statement has Subject and Predicate Attribute / Value (roles).

Value as Occurrence of Attribute in Attribute Occurrence Context.

Context Kind (signature): Subject Kind and Object Kind Attribute / Value (roles).

Subject / Occurrence / Context / Role : Attribute, Value. Concepts. Semiotic Metamodel. Dimensional Encoding: each type as each (pair) kind. Pairs (tags / facets).

Meta Model: Layers Resource relations:

Instance, class, metaclass, occurrence, role. DOM, Actor / Context / Role.

Layer Context: Statement class. Aggregates same Context Statement(s). Next layer metaclass (Occurrence)..

Layer Occurrence: Statement Context metaclass. Aggregates same Context / Occurrence Statement(s). Previous layer context.

Layer Attribute: Statement Context Ocurrence Attribute (occurrence). Previous layer Occurrence.

Layer Value: Statement Context Occurrence Attribute Value (role). Previous layer Attribute.

Layer Aggregation begins with Model initial Statement having a new Context (class) “pushing” previous CSPO right, being the new class the new layer Context and CSP becoming SPO:

(C, S, P, O) : (N, C, S, P).

Functional / Object Oriented Resource API (Model, Statement, Semiotic, Dimensional layers, Meta Resources).

RDF CSPO Quads store backend. RDFS / OWL inference

Backend:

Messages: Events IO / Persistence: Saga Activation / Passivation populating Node local Quad store / persisting peers via DIDs (inference enabled distributed consistency) semantic (resolvable / discoverable) identifiers.

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.

Core API: Model, URI, Resource, Role, Statement, Kind.

Quads Context / Object: class by intension / extension. Transform matches Context signature, filters by Object(s) extension. Resource(s) specification.

Reified Kind(s) / Meta Model. IDs, (Ont)Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings. Resource set specification (SortedSet hierarchies). resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

Deployment / Use Cases:

Purpose driven hypermedia activation Augmented Semantic Content type activation. Messages / gestures. Rules (commands / verbs).

Browser referring context (Work, Peter, Employee). Resource URIs specialized implementations for different connectors / endpoints and content types (DB / OData, REST / HAL, etc.). Feature Resources backends (i.e.: URI for DB interaction).

Purposes: Metamodel declarative goal statement. Fulfill flows (templates / forms: Messages).

Goal: P2P service that connects to services / endpoints (DB, REST, etc.), homogenizes them and exposes an API by which (augmented) knowledge of an stated entity is returned in response (protocol that entails queries / CRUD, object navigation in message / session state contexts). Peer shares / syncs with other peers.

Goal: Intermediate API (HAL for example) aggregating previous objects knowledge (DCI, DOM, OGM, MVC)

Goal: Semantic Browser. Homogenize diverse domains. Query examples. Search session history. Referrer semantics. Collected items in goals roles. Create session purpose document. Link to / from any addressable resource in context / role. Annotate source / destination context roles, attributes

Core (upper / onto) Messages: Getters, setters, nav, etc. Domain Messages: raiseSal: setSal(sal \* increment); promotion: setPosition.

Event sourcing / tracking: married -> marriage occurred. Dataflow: Messages hierarchy. Aggregate contexts from coarse to fine grained  transforms (raiseSal -> setAttr, single - marryWith).

Message dispatch, input statements resolve to applicable messages from switch from behavior to data layer invoking async microservice. Message case matching may involve entering and leaving data, schema and behavior paths if aggregated contexts matches more than one message. Visitor. Execution graph.

Form / Template describing (reified as a Resource in a context model) declaratively subscriptions and actual exchange capabilities (dataflow). Mappings, Transforms.

Processor which acts upon Resource events. Materialize results. Specify declaratively augmentations by means of messages. Metamodel model events I/O, flows.

Hierarchies: layered quad statements are represented by a class hierarchy which root is the Resource<T> monad. There is a subclass relationship between each layer implementing class and the one of the next layer (Dynamic Object Model).

Quads in the context role of lower layers represents occurrences of context enclosing layer.

Assert class hierarchies, order relation (temporal, causal, containment, etc.) by attrs / vals, set / superset relations. TBD.

Metamodel Message event driven I/O protocol:

Discovery: All model kinds are browseable / discoverable. Encode behavior in statements / graph: Comparisons, order. Sort. Order (kinds hierarchy?) Pattern matching, iteration, jumps. Discovery: routes / signatures, next event in bus / graph. Dataflow.

Express Augmentation (Alignment, Activation, Aggregation) as Messages / Transforms. Reified Model entity types / roles (CSPO, Kinds, Layers, etc.).

Example: submitting Behavior layer grammar / context "template" initiates "dialog" for fulfill Behavior expanding Message(s) and nested context layer statements (known / resolvable, new behavior / subitems) needed to complete / update full Behavior layers contexts graph.

Augment. Alignment, Activation, Aggregation Message(s) : Resource set specifications (SortedSet).

Dimensional input set model aggregation specificatíon (from Statement layer, ordered SPOs: order criteria, comparisons. Kinds / class / occurrence / instance order criteria?). Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements layers).

I/O: Augmentations. Basic operation. Resource Set Specification (SortedSet / Statement) matching Model which returns augmented Message response (Model I/O).

## Semiotic Encoding

Subjects: attributes / values. Occurrences: contexts / roles.

(Context, Occurrence, Attribute, Value);

(Context, Sign, Concept, Object);

Metaclass, class, instance, occurrence.

Assert order / hierarchies / relations in dimensional axes. Containment (sets).

Encode IDs: Context Kind, upper (meta) Resources (levels / layers). Resource contents / contexts (identify by occurrences in roles in other contexts, Meta Resources, layers class, metaclass, instance). Compose IDs (hierarchical graph properties encoded string) from outer to inner resources (Context, Kind, Occurrence, Role, Resource). "Operable" IDs (ClassIDs / InstanceIDs: Meta Model reifications / occurrences).

Message parsing (Template, Message Context) matches Form / Flow Augmentation Mappings signatures.

(Template, Context, Attribute, Value); Value as Context: hierarchical models. Same attributes: types / collections.

Semiotic (encodings):

(Context, Sign, Concept, Object);

Object as Sign: Object properties (Concepts).

Semiotic encoding: encode Meta Model (syntax facet) / Functional (Meta Model encoded pragmatic facet) / Dimensional (Meta Model encoded semantic facet) data / reference model (by contexts / upper alignments / aggregation).

Facets contexts semiotic encoding. Facets IO (events) by semiotic encoding of facets input layers.

(Context, Sign, Concept, Object);

Roles encoding: Object as Sign (properties), Sign as Object (types / roles), Concept as Sign / Object / Context, etc.

Grammars encoding: express models contexts layers and augmentation templates in input contexts.

DCI / DOM: Subject, Context, Occurrences, Roles, Attributes, Values, Activation.

Semiotic, Meta Model, Dimensional, Functional DOM. DOM Contexts. DOM Functors. Resources Contexts (CSPO Monads: encoding / addressing).

Activate able DOM Contexts: DOM views (object mappings, etc.).

Map Reduce encoding inputs, grammar templates context mappings. Emit Semiotic reference / data model parent / child properties encoding (Context / SPOs, Context:Subject / POs, etc.).

Model / Semiotic reification.

Facets / Levels / Shapes: Aligned entities / values occurring in aligned models / dimensions.

(Context, Sign, Concept, Object);

Resource Monad / Message Functor (Contexts hierarchy parent). Resource<Context>. Events bindings. Message / Augmentation event declarations / instances.

Order: SortedSet, hierarchies, common upper types rels.

## Normalization

Radix, Digit, Position. SPO: radix:digit. Radix: resource count. Digit: resource radix index. Position: context hierarchy aggregation. Normalized radix encoding: metaclass, class, instance, occurrence context metadata operable IDs. Example: Base 2 normalization, concepts lattice bitstring (routing tables), boolean operations metadata extraction / traversal (contexts masks). IPv6 addresses. Masks (contexts), routes. Address resolution resolves recursively expanded (CSPO hierarchy contexts / occurrences / attributes / values) resources representations (graph statements members nested browseable addresses of ID contexts). Addresses representing behaviors (HATEOAS: browse representation links / addresses renders state / operations, i.e.: address plus according referrer context and representation addresses links states CRUD operations).

Normalization: contextual form IDs. Internal four sides polygon angles for each CSPO addressable IDs. Embed operable meta data in CSPO contextual IDs. Vector space model like. Polynomial coefficients / factors. Adjacency matrix / tensor. Embeddings.

CAM (Ternary bitstring). Routes. CSPO Functional Mappings (contextual / occurrences vector IDs).

Encode IDs: Context Kind, upper (meta) Resources (levels / layers). Resource contents / contexts (identify by occurrences in roles in other contexts, Meta Resources, layers class, metaclass, instance). Compose IDs (hierarchical graph properties encoded string) from outer to inner resources (Context, Kind, Occurrence, Role, Resource). "Operable" IDs (ClassIDs / InstanceIDs: Meta Model reifications / occurrences).

Encode common upper Semiotic / Dimensional Model: Reference Model.

Encode Kind / Context hierarchies.

Encode Augmentation(s) as Resource descriptions (Resource occurrence of Augmentation).

Encode Model(s) as Respurce set. Meta Resources, layers Contexts, Kinds (reified).

Encode Graph Execution Semantics. Dataflow: Context Kind signatures. Iteration, conditional jumps.

Object occurrence of Predicate.

Sets. Quads. SortedSet.

Metaclass / Class / Instance.

Class / Instance ID pairs:

Subject / Context / Role : Attribute, Value. Metamodel. Encoding: each type as each (pair) kind. Pairs.

Semiotic encoding:

(Context, Sign, Concept, Object);

Value as Occurrence of Attribute in Attribute Occurrence Context. Meta Resource context roles).

Encoding: recursive resource quads encoding hierarchy, order, class, instance, attributes. Operate inferences over (upper) patterns (bitstring / lattice). Meta Model, Facets, Levels. Specifications: Signatures, Forms, Flows (encode events / transforms provenance).

Order / Comparison: Ternary truth values results (previous / parent, current / OK, next / children) according Predicates for a Message resource set and and a Mapping comparison (apply augmentation). Octal relations.

## Parsing

Parsing: extract propositions, knowledge assertions (in a domain ontology). Assert propositions links, order, concepts relations (between domains). Link Grammar. ISO TMDM / TMRM.

Parsing: extract prescriptions, knowledge rules (in domain ontology concepts relations: causal, requirements, etc.). StratML.

Parsing: infer possible statements propositions / prescriptions productions. Link Grammar Disjuncts. Embeddings.

Functor / Parser signature:

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

Behavior statements.

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).

# Protocol

Forms, Flows: Meta Model Behavior. Forms / Flows APIs: grammar / semiotic / levels / dimensionally augmented / aware.

## Addressing

Addresses browse state transforms: navigation renders context layers statements transforms with contextual browsing state (IDs, referrer, contextual / occurrences vector IDs, metadata) as parameters. Contexts navigation (Forms, Flows) as functors. Navigation state transforms: possible activations context / argument / attributes / values. Dialog / prompts (argument resources navigation state transforms).

Encoding: nested shapes of recursive cuads (till primitives) identifiers. Patterns / expressions: wildcards, variables, placeholders:

[[123, 456, \_b, $a][\_b][\*][$a]]

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

Siblings: previous / next (Semiotic containment relationship / roles). To do.

Hierarchies: common parent / children (Semiotic containment relationship / roles). To do.

Semiotic model: reification / primitives / upper ontology (meta model, inferred / aggregated). Levels: syntax / grammars, semantics, pragmatics. To do.

Augmentations: addressing (HATEOAS) browsing of: Aggregation, Alignment, Activation transforms (Behavior Messages).

Message: Augmentation (materialised Resource set) specification (functor). Event declaration (I/O patterns bindings). Context layers monad publishes / subscribes (dataflow bindings pipelines).

Ontology matching: signatures encoding. Dimensional ordered aggregated measures. Sets. Semiotic reification.

## Low level

Forms, Flows: Meta Model Behavior / Flow APIs grammar / semiotic / levels / dimensionally augmented / aware. From core model levels to dialog / gestures interaction relations / resources (session level model resources).

Model: RDF Backend.

URIs Services: API for plugging whatever connector may be implemented for behaving in a reactive resource message oriented fashion (backends).

Resource: Abstracts (wraps) URIs Services in a functional API (Resource streams). DOM, Actor / Context / Role (Meta Resources).

Augmentation: Parse Message (event: context quad) according Template (pattern), materialize output Transform. Algorithm (TBD): case classes, pattern matching, destructuring, Resource monad chained operations (Template: functor) functional streams, ADTs.

Dataflow, Reactive: Resource Monad handling of wrapped URIs messages / events I/O via HTTP verbs. Augmentation: Model, Context instance / class (layers), Resources producing / reacting to events. Endpoints: Discovery / Location / Resolution services. URI APIs (signatures discovery).

Meta Graph / Model, Meta Resource(s): Resources / Messages reifying "patterns" on inputs (URI, Resource, Statement, Kind(s), Context, Occurrence, Attribute, Value, Layer Context classes, etc.). Declarative statement for Augmentation shapes applyied to input contexts.

Model Meta Resource: Model components reified Resource types / instances (URIs, Resource, Statement, Context : Layer, Kind, etc.). Augmentation templates "placeholders" (signatures, matching of common upper resources).

Kinds (Application): Basic type inference. Applied over layers CSPO during Activation Augmentation.

Source / Session / Pragma levels. DCI. Data / Information / Knowledge. Syntax, Semantic, Pragmatic. Model state: Context (Resource : data), Kind (Grammar : schema), Dimension (behavior). Context Kind(s) signatures: Dataflow.

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). CQRS. Dialog (EAI pattern: Isis docs).

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 statement CSPO roles.

Alignment: Infer layer missing / deducible attributes and values for CSPO Subjects.

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

Ontology matching: Dataflow: sort statements. Form / Flow Dimensional Metadata. 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.

Functionsl / 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.

## Hypermedia Activation

Addressing (hypermedia):

Content Type: Metaclass

Resource: Class

Representation: Instance

Address: Occurrence / Response (Materialised Message Augmentation)

Request: Message (state flow). Context DOM event API: Request Message Resource in possible domain / range / mapping contexts.

Purpose / content type driven (state / rendering / roles / links /contexts / data / flows / attributes) declarative hypermedia activation application. Model, Application, Domain ontology / upper resources (connectors). Generic API / metamodel (DCI: Form / Flow) client. Extension protocols / APIs. Meta Model levels: from abstract to concrete applications.

Encoding:

Rendering: S: current document URL, P: link tag body, O: href, rel: Context (referrer). Navigation: GET / headers. GET (navigate, possible resources, posible contexts / subjects / attributes): CRUD / Contexts aggregation / transforms / matching. Encoding: CRUD / browse layers (CSPO Patterns Forms / Flows layers de-aggregations / faceted traversals).

Encoding, APIs: REST HATEOAS, JSON-LD, HAL. Distributed (normalized) address ID spaces.

Layers down / up traversal:

C: Anchor rel (referrer);

S: Current URL;

P: Anchor tag body;

O: Anchor href;

Layers up / down traversal:

C: Current URL;

S: Anchor rel / referrer;

P: Anchor tag body;

O: Anchor href;

Activation Protocol:

Protocol: Input statements for querying augmented knowledge (Specification Forms / Flows). Browse result model graphs. Input statements encoding queries / commands: grammars, reified message contexts (templates / forms). Browseable models, contexts, interactions (state / content semantic activation). Dataflow according Messages input signatures.

Dataflow embedding: Resources reifying global state. Specifications: Forms, Flows. Augmentation Dataflow: Functional declarative way of stating Augmentation Transforms over Messages / Resources matching / populated by input Templates performing output Mappings Augmentation reflecting input, model and behavior state.

Source / Grammar / Pragma Levels.

Functional / Dimensional / Semantic Facets.

Reactive Entities: Resource, Model, Message, Kind.

Entities: ID (routes), State (ctx / rel pointers, occurrences). Streams, Dataflow (routes / bindings: addressing).

Transforms, Augmentation (functors / mappings).

Dataflow: Message / Model / Augmentation / Model / Message.

URIs API for annotating network retrievable 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.

Annotate, link, browse resources instances, classes, metaclasses, occurrences in roles in contexts, attributes / values. Services / clients: endpoints: Virtualization (wrapper protocols).

Semantically annotated content types: image/png;face, text/xml;faceImgCoords. RDF schemas describing content, attributes, links in context / target roles. Content types: labels (schemas).

## Facets / Levels

Forms, Flows: Meta Model Behavior / Flow APIs grammar / semiotic / levels / dimensionally augmented. From core model levels to dialog / gestures interaction relations / resources (session level model resources).

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 / Mapping Transform).

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.

Levels: (upper) ontology aggregation layers from primitives until application / domain concepts.

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

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

Model interaction (domain) 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).

Reification: All Model layers and types reified into Resource hierarchy (sub class relationship).

Example: Source, Grammar, Pragma (Session Dialogs / Prompts) levels.

## Features / APIs

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.

Hypermedia Activation. Addressing. Link extended content types resources elements / parts with other resources addressed elements.

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

MDM. Provenance. Versioning. Dimensional context values / queries. Model Facets APIs: Functional, Semiotic, Dimensional Dataflow contexts / order / roles HATEOAS APIs workflows. Protocol: Dialog. Browse / analyze / transform "activations" (REST / JAF) according Facets. Clients / Connectors. OGM.

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).

Backend: RDF(S), OWL, Resource Services (Naming, Index Registry), ISO, Shapes, IDs matching. DIDs Backend.

Runtime: Objects, Events. Distributed Runtime Functional (Dataflow) reactive Resource objects / models (forms / flows).

Deployment: Spring / Vertx / others (Jersey / CDI: Resources Meta Model / Domain APIs). HATEOAS / HAL APIs (Encoding / Dialog Endpoint Protocol). Environment / Container: Messaging / Services. Models I/O.

Integration Connectors: Runtime Model embedded Resources. Sources as reactive stream objects. API.

Integration Clients: Runtime Model embedded Resources. Services as reactive stream objects. API.

Integrations endpoints (augment / extend) Service URIs APIs.

RDBMS.

Rules.

Data Virtualization.

OData.

Solid.

ERP / CRM ESB / EAI.

CMS / Feeds (Hypermedia augmented Resources) / Browser

Messaging (Connectors / Clients)

ML / Big Data.

StratML.

ISO 13250.

ISO 15926.

Dialog: Encoding / Protocol. Activation, Location, Context (Hypermedia type / context interfaces). Augmentations.

Encodings: XML / XSL / XPath / XLink / XForm / XPointer / XQuery (RDFS / OWL / ISO DM / RM Forms / Functors / DOM). Endpoints (streams / signatures): documents (messages) / resources (paths). JSON-LD.

Runtime. Bus. Signatures bindings (reactive), content type / domain context interactions data transforms (roles).

Design: Augmentation (Aggregation, Alignment, Activation Functors), Domain Functors. Encoding. Design Functors Dataflows.

CQRS. Event Sourcing. Functor Commands (HATEOAS browseable APIs).

Domain Driven Development:

DDD Connector / Client / Runtime (Hypermedia / Facade / Dialog Protocol: services / actions). Backend: Augmented Services / Persistence. Representations (resource: types / activation). Domain, Services, View Models.

StratML Client / Connector (Goods, Needs, Products, Goals, Purposes. Exchange ontologies).

ISO Topic Maps / ISO 15926 Facades.

# Features

## Domains schema

Message based Augmentation Events Dataflow. Augmentation Mapping Dataflow allowing to embed merged dynamic state in Model entities (including Mappings Augmentations themselves). Reactive / event driven declarative application framework: Model, meta model, instances as streams sinks / sources (transformations).

## Ontology matching

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).

Determine whether two identifiers refer to the same entity, whether two relations are the same and which results corresponds to instances of the same actions.

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 matching (table, pk, col, val example). Helper upper models for models linking / alignment. Shapes. ISO.

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

Ontology matching: Dataflow: sort statements. Units. Equivalences. Distances / events (order). Services (Augmentation / Context Functors Meta Model mappings / transforms). OntResource; Merged URI(s) wrapper. OntResource hierarchy: layers statement contexts. Facets DOM, Actor / Role. Resource (OntResource Context Roles hierarchies Monad wrapper); Statement : Resource Role quad, Resource.

Ontology matching: Events declarative definition. State change of value in axis in measure of context. Dimensional Model.

Ontology matching (table, pk, col, val example). Helper upper models for models linking / alignment. Event sourcing (“offline” sync). Graph linking / alignment / sinchronization by entailments from event sourcing over inferred state. Reconciliation.

Meta Resource / Models / Messages: IDs / Encoding / Addressing formats. Ontology matching and Template / Augmentation / Transform enrichment (alignments), transforms (functors), materialization (model updates) via Mappings (events) and Meta Resource / Model Encoded Resource declarations (enrich / align, transform, updates algorithms: Encodings).

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. SortedSet hierarchies membership (octal) values.

Ontology Matching: Encode: order, iteration, flows, units, relations, events, enums, etc. Semiotic / Dimensional alignment. TBD.

Ontology Matching. Semiotic. Dimensional. Sets. Functional Reference Model. SortedSet hierarchies membership (octal) values.

Ontology matching. Dataflow: sort statements. 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.

Ontology Matching:

Statements (encoding): Proposiciones / Razonamientos / Cuantificadores / Predicados / Clases / Relacion: validez / valor de verdad (en contextos, ejes, variables: funciones / enunciados / casos de sustitución / equivalencias). Pronombres.

Truth values: Predicate / reasoning. Sets encodings Predicate comparisons matches context templates / transforms mappings: inferences.

Relaciones de equivalencia, clases de equivalencia: reflexividad, transitividad, simetría. Identidad. Propiedades determinan alcanze / relato, dominio / codominio, universo y campo. Clases y relaciones (atributos / valores) determinan matcheos / relaciones entre clases / individuos y relaciones y tuplas / miembros. Formas proposicionales, categorícas (predicados), clases y relaciones (reglas, valores de verdad) para afirmar equivalencias (operar entre relaciones de propiedades).

Relaciones, cardinalidad: (1, n), (n, 1), (1, 1), (n, n). Función / inversa: inyectiva / biyectiva.

Relaciones de orden. Inclusión / jerarquías. Rendering: lattice (encoded statements / properties bitstring / vector). Equivalencias.

Predicate como occurrence de un Predicate: axis (attributes / values). Intensión / extensión (representante partición)

Ontology Matching: Dimensional metaclass / class / instance attributes relationships in axes / contexts / roles evaluations. Resolve equivalence of dimensions, units, measures, values via aggregation of value occurrences. Resolve value kinds by occurrences of Semiotic (grammar) model. Contexts (CSPOs) Attribute / Value "clustering". Sets specifications. Dimensional order relations.

Distributed Inference (dimensional / matching / predictions). Distributed consistency. Event sourcing (Dialog, CQRS Protocol). HATEOAS distributed / lazy reconciliation: hashing of source / destination (hierarchical contextual model quads hashing) state in interactions tokens. Prompts. Distributed Alignment (until reach of goal synchronization states). Distributed state / IDs.

Comparable: parent / child relations (SortedSet). Model sets / dimensional axes alignments.

Semiotic: syntax, semantics (grammar), pragmatics. Verbs: action, passion, state (for / due to action / passion). Models. Dimensional (axis / order) relations. Alignments. (Sorted) Set relations: hierarchical encoding. Verb roles (CSPO / Kinds sets).

Example aggregation: candy (type), red (color attribute), strawberry (flavor attribute).

(dado rango y alcance, universo: U de una relación: P, inferir dominio y codominio, campo: C). TBD.

From Object (O) extension / instances to Context (C) intension / class. Matching grammar shapes. Incrementally render type, role, occurrence context layers.

## Augmentations

Functional declarative way of stating Augmentation Transforms over Messages / Resources matching / populated by input Templates performing output Mappings Augmentation reflecting input, model and behavior state.

Aggregation.

Alignment.

Activation.

Augmentation applied to Context: Aggregation.

Augmentation applied to Subject: Activation.

Augmentation applied to Predicate: Alignment.

Augmentation applied to Object: Role in context.

# Deployment Use Case: Goals Application

Products and Services Exchange Network (PASEN):

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 matching (Data, Schema, Behavior alignments):

Data alignment:  
  
Determine if two instances (example: database records) of two different backends or services refer to the same entity / database row (Customers : John D. / Employees : John Doe).  
  
Schema alignment:  
  
Determine, for example, meaning and equivalences between diverse (aggregated / composite) schemas (equivalent classes / tables, equivalent attributes / columns, equivalent roles / relations).  
  
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).

Purpose driven hypermedia activation:

Protocols / Services / Clients: Context interaction sessions (state flows). Content type activation. Messages / gestures. Rules (commands / verbs). Browser referring context (Work, Peter, Employee).

Use Case: Goals App

Goals App: purpose / goals / domain driven syndication of integrated business / social / cloud application features. User / Groups / Roles Purpose(s), Goal(s), Task(s) "intelligent" tracking oriented focus providing an abstraction and integration layer of players process flows / interactions and players process assets management and semantic orchestration.

Goals App: Semantically annotated gestures / interactions (contexts, purposes messages / interactions / resources / content). Subject context occurrence role attributes values (metaclass, class, instance, occurrences).

Goals App: API Facade for rendering aggregated data roles in contexts interactions topics / subjects assets (conceptual domain contexts axis / state views / activations: Forms / Flows). Example: domain declared Customer (actor / role), Product, Order, Purchase, Invoice, etc. topics / subjects assets rendered in contexts (Sales Report, Expenses Report, etc. embedded / linked dashboards). Wizards.

Goals App: Browse / search / activate: history / relations / referrer context / interaction / gestures roles traceability / (dialogs). Gestures / interactions (actor / asset, actor / actor). Wizards.

Goals App: Hypermedia contents APIs (embedded / embeddable resources: Semantic contextual Wiki / Apache Stanbol / CMS: hypermedia augmentation, knowledge / behavior maps). Integration: augmentation / sync backends / apps. Extension: services / APIs. Annotate / augment link content. DAV protocol (integration / extension facades).

Products And Services 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.

**Monads**

Type constructor (wrapper / type).

Unit wrapper / value instance.

Bind / map / flatMap. Instance argument, transform (static / functor). Instance method (applications).

Functor: A category consists of a collection of nodes (objects) and morphisms (functions). An object could be numbers, strings, urls, customers, or any other way you wish to organize like-things. (X, Y, and Z in the graphic are the objects.).

A map is a function to convert something from one object to another. (f, g, and fog are the maps). Google tip: A map between objects is called a Morphism.

So an object could be simple like a Number or a String. An object could also be more abstract like a Username, A User API URL, User API HTTP Request, User API Response, User API Response JSON. Then we can create maps or morphisms between each object to get the data we want.

Examples of morphisms: Username -> User API UrlUser API Url -> User API HTTP RequestUser API HTTP Request -> User API ResponseUser API Response -> User API Response JSON

Google tip: Function Composition is a way to combining multiple map or morphisms to create new maps. Using Function Composition we could create a map from Username directly to User API Response JSON.

Now that we understand what it means to be Mappable, we can finally understand what a Functor is.

A Functor is something that is Mappable or something that can be mapped between objects in a Category.

An Array is Mappable, so it is a Functor. In this example I am taking an Array of Numbers and morphing it into an Array of Strings.

Note: One of the properties of a Functor is that they always stay that same type of Functor. You can morph an Array containing Strings to Numbers or any other object, but the map will ensure that it will always be an Array. You cannot map an Array of Number to just a Number.

We can extend this Mappable usefulness to other objects too! Let's take this simple example of a Thing.

If we wanted to make Thing mappable in the same way that Array is mappable, all we have to do is give it a map(morphism) function. And that is a Functor! It really is just that simple. Google tip: The "Thing" Functor we created is known as Identity.

Monad:

Sometimes functions return a value already wrapped. This could be inconvenient to use with a Functor because it will re-wrap the Functor in another Functor.

This is where flatMap comes in handy. It's similar to map, except the morphism is also expected to perform the work of wrapping the value.

Summary:

A Functor is something that is Mappable or something that can be mapped between objects in a Category.

A Monad is similar to a Functor, but is Flat Mappable between Categories.

flatMap is similar to map, but yields control of the wrapping of the return type to the mapping function.

Monads:

Monads are a way to compose type lifting functions: g: a => M(b), f: b => M(c). To accomplish this, monads must flatten M(b) to b before applying f(). In other words, functors are things you can map over. Monads are things you can flatMap over:

A monad is a way of composing functions that require context in addition to the return value, such as computation, branching, or I/O. Monads type lift, flatten and map so that the types line up for lifting functions a => M(b), making them composable. It's a mapping from some type a to some type b along with some computational context, hidden in the implementation details of lift, flatten, and map:

Functions map: a => b which lets you compose functions of type a => b

Functors map with context: Functor(a) => Functor(b), which lets you compose functions F(a) => F(b)

Monads flatten and map with context: Monad(Monad(a)) => Monad(b), which lets you compose lifting functions a => F(b)

Map means, “apply a function to an a and return a b". Given some input, return some output.

Context is the computational detail of the monad’s composition (including lift, flatten, and map). The Functor/Monad API and its workings supply the context which allows you to compose the monad with the rest of the application.

The point of functors and monads is to abstract that context away so we don’t have to worry about it while we’re composing things. Mapping inside the context means that you apply a function from a => b to the value inside the context, and return a new value b wrapped inside the same kind of context.

Observables on the left? Observables on the right:

Observable(a) => Observable(b).

Arrays on the left side? Arrays on the right side:

Array(a) => Array(b).

Type lift means to lift a type into a context, blessing the value with an API that you can use to compute from that value, trigger contextual computations, etc… a => F(a) (Monads are a kind of functor).

Flatten means unwrap the value from the context. F(a) => a.

Dataflow, Reactive: Function composition creates function pipelines that your data flows through. You put some input in the first stage of the pipeline, and some data pops out of the last stage of the pipeline, transformed. But for that to work, each stage of the pipeline must be expecting the data type that the previous stage returns.

A monad is based on a simple symmetry A way to wrap a value into a context, and a way to unwrap the value from the context:

Lift/Unit: A type lift from some type into the monad context: a => M(a)

Flatten/Join: Unwrapping the type from the context: M(a) => a

And since monads are also functors, they can also map:

Map: Map with context preserved: M(a) -> M(b)

Combine flatten with map, and you get chain — function composition for lifting functions, aka Kleisli composition:

FlatMap/Chain Flatten + map: M(M(a)) => M(b)

Monads must satisfy three laws (axioms), collectively known as the monad laws:

Left identity: unit(x).chain(f) ==== f(x)

Right identity:[m.chain](about:blank)(unit) ==== m

Associativity:[m.chain](about:blank)(f).chain(g) ==== [m.chain](http://m.chain/)(x => f(x).chain(g)

Functor: mapping between categories A / B using a function. Map, alcance, rango, dominio, imagen (infer, aggregate). Connections: same number of items (inyective, biyective). Function: de wrapper en wrapper (morphism new image). Message / Augmentation Metamodel reifications.

Actor / Role. Dynamic Object Model (OGM / Kinds). Golden Braid: Metaclass, class, instance, occurrence relation relative to layer context levels.

Reference Model (DOM / Actor / Role OGM):

Type : Value;

(Value, Type);

(Type, Type);

(Value, Value);

Aggregation: cons cells tree traversal. Layers contexts representation. Dataflow augments updates / append corresponding tree cells: (first: (rest: nil).

(C (S (P (O))));

Aggregate / Augment inputs / transforms: Parsed model streams. Reactive data structures.

Encoding: addressing, semantic graph networks. URN overlay semantic addresses encoded mappings.

Dataflow:

Input statements: Augment. Aggregate, build layers contexts representation quads. Parse:

Monadic parser combinator: aggregated metamodel nested layers contexts corresponding wrapper / wrapper hierarchy types DOM / AST (from contexts quads aggregation nesting hierarchy levels).

Augment: signatures (case meta / classes / instance / occurrences) dataflow. Apply input statements: new inputs / kind new attributes (person, employee -> position, salary). Parse.

Protocol: Forms / Flows augmented / parsed representation / metamodel I/O. HAL / HATEOAS endpoint encoding for navigation, transforms and inputs augmentation.

Protocol: Forms / Flows Dialog / Prompts resolution. Context roles, wrapper kinds navigation / transforms declaratively stated in encoded representations.

Monad: context type. Metaclass.

Monad: wrapped type. Class.

Monad: wrapped value. Instance.

Monad: wrappers hierarchy context type instance. Occurrence.

Unit. Type constructor (hierarchy context / member types / values factories). flatMap / map / flatMapN.

Monad: context type case classes hierarchy? Factory methods, hierarchy types / members values / signatures (wrapped context / values) cases (predicates):

Optional: None | T [case context | case context / case value signature]

Result: Error | OK [case context | case context / case value signature]

Writer: Value, List<S> trace

Monads: Quads Contexts wrapper of Occurrence / Subject aggregations. Root type: Resource. Kind (model / semiotic / domain meta resources) functors filter / traversal (streams: flatMap Resource set specifications).

Wrapped resources holds references to its wrapping occurrence contexts (Resource root type interface, contexts hierarchy / reification levels: upper layer instances reify layer occurrences in contexts).

Wrapping unit / bind: layers traversal until container wrapped category type is met. Example: Behavior(s) of an Entity. Augmentations (types, roles, attributes reified model / domains aggregations). Example: Model, SK, P; Domain, someSubjKind, somePredicate.

Form / Flow: state navigation / browse. Dialogs / Prompts (contexts). Messages I/O: Augmentation signatures (kinds) streams.

Messages: Augmentation according signatures. Broadcast. Key / Value, Event sourcing routes (levels). Form / Flow Dialogs / Prompts Messages  for signatures (kinds) of unknown Resources resolution (layers wrapping traversal, CSPOs layers resolution for CSPO signatures / attributes / values).

Encoding: IDs. Embed metaclass, class, instance, occurrence metadata (context, role, attributes, values). Functional APIs. Wrappers / Transforms (augment: aggregate / classify, roles, properties "graph" rels). Polygon Vector Space Model. ANN embeddings / autoencoders.

Forms / Flows Dialogs / Contexts. Protocol. Resources, addressing, representations, navigation / traversal: properties "graph" rels (Wrappers / Transforms). Functional APIs.

Sets encoding: properties in axes (kinds). SortedSet (hierarchies). Metaclass, class, instance, occurrence properties in axes for CSPO IDs. Augmentations: property graph rels navigation / traversal. Dialog Forms / Flows "state" contexts.

Contexts Wrappers kinds Transforms / Traversals functors: Augmentations declaratively stated in upper Context layers (kind classification, kind roles, kind attributes / values).

Dialog Forms / Flows "state" Contexts browsing (upper Context SPO kinds: current context streams).

Augmentation navigation of Transforms / Traversals as a Context (streams / filters). Levels / reification.

Order. Iteration. Predicates (resource meta / domain / kinds). Streams filter, conditionals, jumps. Aggregation. Functional mapping / reduce, etc. Dataflow signatures (case meta / classes / instances /occurrences)

Resource API (layer roles):

Statement: Resource (CSPO: Statement) Property graph URL wrapper. URL occurrences aggregate. Functional occurrences properties (roles / streams) for Statement wrapped URL:

Contexts: Metaclass URL occurrences.

Subjects: Class URL occurrences.

Predicates: Instance URL occurrences.

Objects: Occurrence URL occurrences.

Statement wrapped URL occurrences functional roles Kinds: Map<Role, Kind> (reified in Metamodel).

CSPO Kinds (streams functors) declaratively stated in aggregated layers CSPO occurrences (kind classification, kind roles, kind attributes / values).

Augmentation navigation of kinds Transforms / Traversals as streams / roles (filters). Levels / reification: kinds from Statement / roles layers (reification / levels axes).

Query / stream context selectors: ID (URL), layer context type, layer context role, layers kinds. Transforms (functor kinds: augment / browse query context according kind specification with corresponding statements).

Levels: Grammars (kinds functors dataflow signatures). Productions: Augmentations (parsed / produced in navigation contexts). Dataflow order (sets / hierarchies).

Aggregation: inputs in contexts layers. Case matching: metaclass, class, instance, occurrence, kind grammars.

Parsing: Monadic combinator parsers: quads contexts layers (recursion). Metaclass, class, instance, occurrence, kind aggregations parsers.

Augmentation. Functional API: Monadic DOM / AST Parse Tree (cons cells) context layers hierarchy wrappers, Resource metaclass, class, instance, occurrence, kind hierarchy wrappers (i.e: contexts instances / parsed kinds).

Dataflow: Aggregation, Parsing, Augmentation. Streams: reactive / event driven. Model reified Message Functors / Transforms.

Zippers: Aggregation / Addressing: Locations / Contexts. Parsing. Monads. Augmentation (navigation / transforms) Reactive Streams (location observers / observables: paths / kind paths dataflow signatures).

Inputs / CRUD navigation / transforms Augmentation Dataflow: parsing / zippers over core Model reified context layers meta Resources AST / DOM parsers (zippers paths in meta Resource aggregates / parses inputs in contexts). Resource metaclass, class, instance, occurrence paths / cases matching zippers.

Blockchain: P2P (JXTA / Git) DL Distributed Ledger inter Node Backend. DIDs (traceable semantic state: Distributed IDs encoded / embedding result of transforms, labeled / property graph statements / contexts: saga / zippers). Smart contracts (signatures: Dataflow). Monads (immutable state, transactions: functor morphisms). Zippers (chain contexts: mutable chain branches, dimensional contexts / labeled property graphs). Reactive Augmentation (I/O) APIs: Resource metaclass, class, instance, occurrence paths / cases case matching (inputs quads, parsed DOM, outputs quads). Quads Forms / Flows Protocols.

Node Protocol: Forms / Flows DAV HAL / HATEOAS Client Application Sessions (navigation contexts). JCR, Hierarchical structures (XML, XPath, XSL, XLink, XQuery, XPointer) representations of augmented reactive DOMs. Representation Levels (onto meta resources): metamodel / session / domains. Behavior encoded in (augmented) representations functional contexts traversals. JXTA / Git Backend inter Node P2P Blockchain Node quads DL IO sync. Connector Nodes: reactive dataflow (signatures: smart contracts).

Augmentation

Aggregation, Alignment, Activation:

Class (relation) and instance (relationship) being the things that could be asserted for each (domain / range for classes, pairs of "roles" for instances and attributes for both: as property graph) the difference between relation and relationship.

A naive approach of render this in pseudo RDF / RDFS:

Marriage : Relation;

Husband domain Marriage;

Husband range Male;

Wife domain Marriage;

Wife range Female;

Marriage properties (date, etc.);

aMarriage : Marriage;

aMarriage husband Pete;

aMarriage wife Mary;

Marriage attributes (domain / range). Reified Relation instances entails statements (expands links, attributes in property graphs) for Relationship roles / players attributes:

Peter marriedWith / husbandOf Mary; domain: spouse / husband; range: spouse / wife;

Mary marriedWith / wifeOf Peter; domain: spouse / wife; range: spouse / husband;

marriedWith / husbandOf / wifeOf statements in a CSPO context: aMarriage;

There should be an inference method materializing inferences of role instances attributes according the Relation class Relationship instance roles they play.

Or, if RDF Quads are not available, entailed properties schema / instances: marriedWith:\_0 / husbandOf:\_0 / wifeOf:\_0 instances of corresponding relation class attributes. Entailing relationship (aMarriage) instances context attribute.

marriedWith:\_0 rdfs:type / rdfs:subPropertyOf marriedWith (expansion property kind).

The case is that a "terminals" relation relationship resource statements "expansion" materialized view renders the Relationship "extension". Way back entailing / inferring relation / relationship class / attributes roles should be possible.

Source (higher order) relations may relate relations / relationships with other relations / relationships thus allowing a richer set of concepts into an ontology / dialect. Example: Peter / Mary Husbandhood related to their Marriage. RDFS domain / range properties provides the inference means here to parse such a relationship entailing relation context / roles.

Rules for expansion: if a Relation is a class for Relationship(s) which has Role(s) for Resource(s) in SPO statements the statements expansion is the "materialized" view of the Relation instance in SPO statements.

Having a tuple:

(Template, Relation, Relationship, Role, SPORole, Resource);

Aggregated Template SPORole Resource should enable the use of some query mechanism (SPARQL? Zippers?) for building output triples. Aggregation intension / extension bidi transform.

Reify from lower layers to expanded statements materialized views and aggregate forward (I/O, Augmentation).

Transform: apply Kind Relation. Relation defined by extension (tuples) and intension (property / attributes relations).

Layers hierarchy:

(Context, Object, Concept / Sign, Value);

(Resource, Resource, Resource, Resource);

(Role, Resource, Resource / Attribute, Resource / Value);

(Statement, Role, Resource, Resource / Attribute);

(Entity, Statement, Role, Resource);

(Template, Entity, Statement, Role);

(Mapping, Template, Entity, Statement);

(Flow / Augmentation, Mapping, Template, Entity);

(Behavior / Message, Flow, Mapping, Template);

(Measure, Behavior, Flow, Mapping);

(Unit, Measure, Behavior, Flow);

(Dimension / Axis, Unit, Measure, Behavior);

(Value, Dimension, Unit, Measure);

(Concept, Value, Dimension, Unit);

(Object, Concept, Value, Dimension);

(Context, Object, Concept, Value);

Reify from lower layers to expanded statements Resource materialized views and aggregate forward into relations / relationships: contexts (I/O, Augmentation).

(husband, role, resource, ?);

(aMarriage, husband, role, resource);

(Marriage, aMarriage, husband, role);

Relation / role type promotion. Contexts. Augmentations (of promoted players role kinds transforms): relationship and expanded members / attributes / links / relations.

Relation<Relationship<C, S, P, O>> (CSPO : Relation) Monads root hierarchy.

Dataflow:

Monads / Zippers (cons / graphs). Aggregation, recursion. Expressions. Signatures.

Aggregation: nesting. Relationship C Relation holding same C context role corresponding / prefix of aggregated SPOs, same CSs for aggregated POs, etc.

Relationship: Kinds / Roles. Aggregations: traversal / expressions (bound functions renders CK, SK, PK, OK).

Parent layer: current layer extension / expansion.

Current layer: C intension, O extension.

Next layer: current layer intension.

Dataflow: perform augmentations on layers instantiations. Observers, observables, signatures.

Inferir relación dominio / rango, alcance / campo. Describir relacion n-aria como predicados.

TBD.

Model Hierarchy:

Resource : Relation. Relationship CSPO: metaclass, class, instance, occurrence Resource Relation context roles.

Relationship CSPO: Context, Kind, Statement, Role, Resource reified Relation(s): Resource Relationship(s) instances (aggregation). Metaclass, class, instance, occurrence.

Dimensional: Events. Causal, roles (marriage). State, predicate properties from / to (single / married: marriage, married / single: divorce). Actor / Class / Role: metaclass, class, instance, occurrence. Marital status example.

Relation by expressions / predicates: brotherhood(a.parent = b.parent). Predicates linking (actor / class / role) to a dimensional event.

Reify Attributes / Values as Relations (Relationship Kinds instances).

ISO:

About Relationship: many Relationship instances are the result of an Activity, e.g. Marrying – Marriage, Assembling – Assembly, Containing – Containment, Connecting – Connection, Employing – Employment, etc.

We model that by typing a Relationship with a (meta) ClassOfRelationshipWithSignature that is defined as a ClassOfRelationshipWithSignature is a ClassOfRelationship that may have a RoleAndDomain specified for each end. (where RoleAndDomain simply stands for ‘a Class in a Role’)

The instance of ClassOfActivity CONNECTING-A-TRAIN

:CONNECTING-A-TRAIN rdf:type dm:ClassOfActivity .

:CONNECTING-A-TRAIN :hasPartiipant1 RoleAndDomain1 .

:CONNECTING-A-TRAIN :hasPartiipant2 RoleAndDomain2 .

:RoleAndDomain1 rdfs:subClassOf rdl:LOCOMOTIVE .

:RoleAndDomain1 rdfs:subClassOf rdl:PULLER .

:RoleAndDomain2 rdfs:subClassOf rdl:TRAIN WAGON .

:RoleAndDomain2 rdfs:subClassOf rdl:PULLED .

The instance of ClassOfRelationshipWithSignature

:CONNECTION-OF-A-TRAIN rdf:type dm:ClassOfRelationshipWithSignature .

CONNECTION-OF-A-TRAIN :hasClassOfEnd1 RoleAndDomain1 .

:CONNECTION-OF-A-TRAIN :hasClassOfEnd2 RoleAndDomain2 .

The typed Relationship

:myRelationship rdf:type tpl:CONNECTION-OF-A-TRAIN ;

:myRelationship :hasPuller myLocomotive ;

:myRelationship :hasPulled myTrainWagon .

An instance of Relationship, typed with this metaclass CONNECTION-OF-TRAIN, can be linked to an instance of Activity, typed with ClassOfActivity CONNECTING-A-TRAIN, with an instance of above CauseOfEvent.

When I connect a train I cause the Event ‘train is connected’, which leads to a state that the locomotive and the trainwagon instances are connected, a fact that is recorded with an instance of ConnectionOfTrain relationship.

Dataflow:

Data / Information / Knowledge. Levels. Formalization. Reference Model. Transforms:

Aggregate / Deaggregate CSPO (expand / collapse intension / extension) dataflow. Layers in / out traversal:

Class (prev): (Class, Subject, Property, Object);

Instance (this): (Value, Context, Concept, Object);

Occurrence (next): (Value, Dimension, Unit, Measure);

Levels: aggregate dimensional context properties. Relation levels: data / info / knowledge expansion. From DCI / actor role / ontology "use cases" (rendered "real world" application "behavior") to fine grained Resources CSPO Statements.

Data: product price / marital status.

Information: price variation / state change.

Knowledge: increase, decrease / marriage, divorce.

TBD.

Relationships: verbs (infinitive), relation: verb (conjugation, CSPO context roles). Verb: action, passion, state (of roles).

(C[context;], ,S[action; role: schema; player: data], P[state; verb: domain/range; mappings: properties], O[passion; role: schema; player: data]);

(data:schema, behavior:state, schema:data);

Graph Normalization (kinds). Levels. Direction (labels): fatherOf, sonOf (property types: inverse of, reflexive, simetric, transitive, etc.). Reification (kinds type properties / values implies relationship instance (salary: Employment, same sonOf value: brotherhood), relationship instance implies kinds attributes / values).

Dimensional: from possible knowledge to information to data. From actual data to possible information to knowledge. Order: from types promotion (domain / range).

Knowledge: Functor: Ownership (Person : owner, Dog : owned). Relationship

Information: Morphism: Owns (Peter, Fido). Relation (anOwnership). Relationship roles promotion. State change / events (reified relation state).

Data: :ownsDog, :owner. Attributes / Values. State: to / from relationship relation attributes / values).

(Knowledge, Information, Attribute, Value);

Objective: achieve normalized form from which aggregate data into layered (data / information / knowledge) layer occurrences relationship (layer roles relations) and enable further knowledge to be de aggregated into its information and facts. Layers specialization in each part of the model.

Dataflow: Reactive DOM. Events. Augmentations. Streaming I/O (signatures, domain / range roles ordered pipelines selectors). Model events bus.

Resources (top layer input): populate / augment / align lower context layers (metamodel / upper onto / kinds placeholders). Render Statements, Roles, Entities, Interactions, Contexts / Mappings, Flows / Behaviors, Dimensional and Semiotic aggregations to be populated / augmented via further input facts (Grammar).

Dimensional / Context (bottom layer input): base upper ontology resources / browse / roles prompts till base Resource layer (facts).

Metamodel subscriptions: Reified (static) layers contexts observes / observable of upper layers types, observer / observable of lower layers types Model events. Matching instatiate contexts.

Domain subscriptions: Context layers (instances) observes / observable of upper layers instances, observes / observable of lower layers instances Model events. Matching augments contexts.

Example: Data (relation / facts: Entity), Information (events: Template / Interaction), Knowledge (relationship: Mapping / Context) layers abstractions.

Browse: Form / Flow. Selectors / zippers. CRUD (HATEOAS / HAL APIs).

Browse Streams Dataflow: Data / Information / Knowlege streams. Upper: SP/CS selector (intension contexts statements) expands lower layer information / facts. Lower: CS/SP selector (extension object statements) aggregates upper layer information / knowledge.

Dataflow: Metamodel (reified / static) contexts subscriptions. Contexts monads hierarchy (Relationship).

Dataflow: Model (domains / instances) contexts subscriptions. Contexts types hierarchy (Relation).

Inputs: Facts (data). Aggregate. Dataflow events.

Inputs: Relationship (knowledge). Populate facts (roles prompts). Dataflow events.

Browse: expand facts (till input data).

Browse: aggregate relationships (till aggregated knowledge).

Browse Dataflow: Selectors. Available roles input kinds (apply lower layer relationship). HATEOAS. Transforms / mappings / functions / contexts as functions.

Input Dataflow: Apply Kinds. Role promotion. Knowledge input aggregation. Data facts prompts. CSPO Dataflow: monadic functions transform pipelines (materialize / update role / kind knowledge / data statements).

Browse / Input Dataflow: Context. REST HATEOAS state browse / render / submission. Interactive "dialogs". Form / Flow APIs.

Levels: contexts hierarchy polymorphism.

Metamodel:

Encoding. Context selectors (location / dataflow):

ID:Occurrence (CtxClass [metaclass, relationship context roles / Context CSPO Kinds. ID:Occurrence], TypeClass [class, relation roles. Match data with kinds. ID:Occurrence], TypeInst [instance, input / prompts roles facts / data event. ID:Occurrence]);

Resource : (ID: URL, Occurrence: Resource); Reified Resources, Roles, Contexts.

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

Relation (Resource, Resource, Resource, Resource) : Resource;

Relation instances TypeClass hierarchy (RoleRel, StatementRel, etc.).

Relationship (Role, Role, Role, Role) : Relation;

CtxClass monadic wrappers hierarchy (RoleCtx, StatementCtx, etc.) wraps corresponding Relation hierarchy types.

<https://wiki.haskell.org/Zipper>

<https://wiki.haskell.org/Zipper_monad>

<http://learnyouahaskell.com/zippers>

<https://www.functionaljava.org/features.html>

<https://stackoverflow.com/questions/5919901/data-structure-differentiation-intuition-building>

<https://www.google.com/search?q=The+Derivative+of+a+Regular+Type+is+its+Type+of+One-Hole+Contexts&oq=The+Derivative+of+a+Regular+Type+is+its+Type+of+One-Hole+Contexts>

<https://github.com/jon-hanson/parsecj/blob/master/README.md>

<https://blogs.msdn.microsoft.com/lukeh/2007/08/19/monadic-parser-combinators-using-c-3-0/>

<https://www.w3.org/Data/events/data-ws-2019/>

<https://dzone.com/articles/functor-and-monad-examples-in-plain-java>

<https://dzone.com/articles/whats-wrong-java-8-part-iv>

<https://curiosity-driven.org/monads-in-javascript>

<https://hackernoon.com/functional-javascript-functors-monads-and-promises-679ce2ab8abe>

<https://medium.com/javascript-scene/javascript-monads-made-simple-7856be57bfe8>

<https://importantshock.wordpress.com/2009/01/18/jquery-is-a-monad/amp/>

<https://blogs.msdn.microsoft.com/lukeh/2007/08/19/monadic-parser-combinators-using-c-3-0/>

<https://github.com/jon-hanson/parsecj/blob/master/README.md>

<https://dzone.com/articles/parsing-in-java-part-3-diving-into-peg-parsers>

**Meta Model**

Meta Model:

Arrangement of layered typed CSPO quads statements in which each CSPO role / type plays the role of "aggregating" previous layer abstract knowledge into more concrete aggregated contexts (statements) instances until a "reference model" CSPO type arrangement is achieved, which is the root of all other layers hierarchy.

The purpose of this is to achieve some ontology matching capabilities over an upper abstraction set of layers and to enable a Functional Knowledge Base Interaction APIs integration / virtualization overlay for matching and consumption of distributed datasets via endpoints dataflows.

**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.

**Matching / Relations / Attributes:**

One of the intentions of having all this layered infraestructure is to be able to inspect "relations", being them "reified" into a Relationship construct, or being them single attributes and values for a subject enabling the possibility of "align" one into another for ontology matching purposes.

(a, b, c: Kind, Role, Statement): Reified Rel. to / from expanded Attributes / Values. Matching / roles (intension / extension).

Context DOM: parent / child; previous / next siblings; attribute / value (determined by CSPO roles). Class / instance DOM relation for parent / children layers instances.

**Ontologies:**

Context layers instances. Levels. Example: Dimensional ontology. Ontologies should be able to be built upon Reference Model layer CSPO types arrangements.

Dimension, Unit, Measure, Value.

Axis, Behavior, Flow (state change), etc.

Primitives: dimensional upper ontology. In / Out, Prev / Next, Pick / Drop, etc. Opposites. State change (current). Events, state flows. Marriage example.

**Message Dataflow:**

Relationship, Mapping, Relation streams / signatures. Messages: Context instances. Functional Knowledge Base Interaction APIs.

Aggregation: Browse / Transform.

Alignment: Inference.

Activation: Dataflow type (signatures). Message dispatch (domain / range ordered). Aggregation.

**Relations:**

Inference. Relation types: transitive, reflexive, simetric. Campo, alcance, dominio, rango, transform / function: infer / aggregate. Context functor / monad.

Inputs: (Context / Relation, PK, column, value);

Inputs: (Infer S Kind / Role, S, P, O);

Inputs: aggregate occurrences. Statement Context for each SPO as Occurrence with corresponding Attribute / Value (S: PO, O: SP, P: SO, etc.).

Inputs (infer rels): Part / Whole. SPO / OPS. Attribute / Value.

Inputs (infer rels): Containment. SPO / SPO. Parent / Children. Occurrences of Contexts of same Context layers (inherited Contexts). Example: (Mapping, Mapping) for (Relationship, Mapping). Super / Sub type Contexts instances relationships.

Inputs (infer rels): Order. SO Ps Domain / Range.

Input (infer rels): Event. Prev / Next state change. Type promotion.

**Reactive Functional Reified Metamodel:**

Transforms: Match Selectors. Hierarchy polymorphism. Contexts streams. Browse Metamodel. Context, Subject Selectors.

Transforms: Templates. Context instances (CSs) declaration / augmentation (POs) Selectors. Metamodel activation. Predicate / Object Selectors.

Encode Match / Template as Context. CS: Match, PO: Template Selectors. Apply Templates (role bindings / prompts) declaration / augmentation activation to matching selected CS streams.

Augmentation: Aggregation, Alignment, Activation Reified Match / Template dataflows. Reactive Model instances Match / Template dataflows.

Selectors:

Apply Role to Statement : Statement / Statement to Role : Role.

Apply Kind to Role : Role / Role to Kind : Kind.

Apply Relation to Kind : Kind / Kind to Relation : Relation.

Apply Mapping to Relation : Relation / Relation to Mapping : Mapping.

Apply Relationship to Mapping : Mapping / Mapping to Relationship : Relationship.

Apply Context to Relationship : Relationship / Relationship to Context : Context.

**Meta Model / Backend:**

Reference Model / Occurrences annotations matrix:

Attachment: show an example of a fully expanded set of last model layer (Relation) and its corresponding occurrences property graph annotations. Previous model layers may be annotated accordingly in respect to their ability to aggregate more abstract contexts (Resource, Context) properties.

Reference Model / Occurrences annotations matrix:

Lattice / FCA: Contexts / Resources. Objects / Attributes (Contexts instances axes). (X, Y): Z (for corresponding pair types / functional transforms).

Statements and annotations: FCA Lattice / FCA Contexts / Attributes. Objects / Attributes (Contexts instances axes). (X, Y): Z (for corresponding pair types / functional transforms). Typed calculus in FCA context development. Sets.

IDs / bitstring encodings. Algebraic / arithmetic activation flows / templates / transforms metadata encoded selectors. Vector Space Model quads polygon embeddings.

Interfaces:

ID : URL;

Context: ID;

Object : Context;

Sign : Object;

Value : Sign;

Labeled Property Graph annotations example. Augments Reference Model. Statement example, Statement context aggregates SPO annotations (Statement occurrences data in other Reference Model layers contexts):

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

Resource : Value (Resource, Resource, Resource, Resource);

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

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

Kind : Role (Statement, Role, Kind, Resource);

Relation : Kind (Statement, Role, Kind, Relation);

Relationship : Relation (Relationship, Role, Kind, Relation); \*

\*: Relationship: Aggregated Relation Statement Relation (Object) Roles / Kinds.

Hierarchy: render / process layers (Relation, i.e.) as Context, Resource, Statement, Role, etc. (upper layers) contexts (i.e.: reify Relation as Context, Resource, Statement, Role, Kind layer contexts). Context semiotic layer: aggregation, ontology matching.

Intension / Extension: S / O.

sub / super hiers, containment: P / O.

Relation reification: Relation statement object: relation instance. a: Role / b: Kind: relation ends. (a): Role reifying rel attrs / values. (b): Kind Resource reifying rel subject (rel players).

**Model:**

Message Events Bus.

Context Monad & type hierarchy (AST). CSPO parameterized types & aggregation (layers hierarchies specializations).

DOM:

Layer (Contexts): events producer / consumer (streams observer / observable).

DOM Parsing MetaModel: aggregate occurrences containment / hierarchies. Layers: subtype / supertype browse parent / children / siblings (order) and Attributes / Values relations.

Context::matchFilter(arg : Context): signatures / kinds stream predicate.

Context::applyMap(arg : Context): apply updates (CS Contexts / PO CRUD) matching filter predicate Context. Fires event bus messages.

Matching applies to meta-model signatures (internal aggregation, alignment and activation augmentations) and to domain / actual models signatures. Render new Attributes / Values and CSPO statements.

DDD: Declarative AST / Dataflow VM. Runtime. Encodings (layers / messages / activations). Event sourcing backends (Blockchain). Patterns (DCI).

RDFS / OWL / Graph Backend: Reified Metamodel. Labeled Property Graph. APIs.

Functional Meta Model Context Layers. Reactive streams. Match / Template Selectors. API.

Stream Resources Connectors (P2P Connector Bundles Context I/O). Dataflow. Backends / Augmentations / Endpoints model layers. APIs (reactive / events).

Layered abstraction levels streams options / menu semantics (REST). Forms / Flows. Browse, match / transforms high level APIs. HATEOAS CRUD / Flows.

Patterns. Input formats. Sample data.

**FCA / Concept Lattices**

Model context statements of an upper ontology and occurrence statements for each ontology concept occurrences using FCA (Formal Concept Analysis) and rules from a pseudo-grammar.

Context statements / Occurrence statements: Grammars. Concepts / objects hierarchies: CSPO statements concept types / kind rules / terminal instances. Productions: concept types / kind rules / terminal instances mappings / flows.

Aggregate kind rules (grammar) into context statements / Occurrence statements: polymorphically, Kind context applies to all Resource hierarchy (all lattices).

Contexts parsing: monadic parser combinators / monadic AST. Recognize context types from (surrounding) reified kind types / rules (link grammar). TBD.

Rules are of the form:

(TypeA, TypeB) > AggregatedKindResources;

For example, in Relation lattice:

(someRoleA, someKindB) > AggregatedRelationResources;

Dispatch: Model (lattices) observer / observable (streams / functional) of grammar (contexts case match) events.

Lattice (FCA Contexts) population and Augmentation:

Base layer: Context. Resource, Statement, Role, Kind, Relation layers.

Input layer: case match layer grammar type signature. Reactive dataflow dispatch inputs top-down / bottom-up contexts layers hierarchy (lower hierarchy layers polymorphically materializes upper hierarchy layers) for Augmentation.

Lower hierarchy layers contexts mapping / function transforms into next upper layer context (example: Roles to Kinds). TBD.

Contexts (TBD):

Data layer: Resource, Statement, Role, Kind, Relation contexts (aggregation).

Information layer (occurrences / interactions): Data layer contexts statements products as new contexts (aggregation).

Knowledge layer (Dimensional / DCI Contexts): Information layer contexts statements products as new contexts (aggregation).

Tensor like arrangements by FCA / grammars (TBD):

Aggregations: Data layer contexts containing Information layer contexts containing Knowledge layer contexts. Encoding: embeddings, object / attribute features bitstrings, VSM (Vector Space Model) quad polygon angles. Order encoding (lattice). Activation / Aggregation (CUD): preserve order (lattice), arrange input message as production of corresponding context shape. Retrieval / transforms: resolve production of corresponding context shape for input message.

Augmentation:

Activation: Layer receives matching context message.

Aggregation: Layer resolves productions of grammar rules for context message.

Alignment: Resolution of relevant knowledge and input message. Emits output message.

Notes:

Context occurrences statements (i.e.: Statement in Kind context). Occurrence contexts: (S, S); (P, P); (O, O);

FCA: Lattice. Ordered Sets. Intension / Extension.

To Do:

Model Order. Axes: dimensions, units, measures. Events (measures / relations / state boundaries in data / information / knowledge levels: price, price at moment in time, variation, tendence predictions, idem for distances, etc.).

Hierarchies: metaclass / role, class, instance, occurrence (parent, children, previous, next, attribute, value). Encoding. Comparisons. Functional traversal (streams).

Dimensional contexts: Contexts from Occurrences contexts statements. Dimensional contexts: Events (attributes). Order relations assertions by context occurrences hierarchy domain / range, set / superset attributes relations.

(Mapping, Kind, Role, Statement);

Event (Dimensional context attributes): (Mapping / unit / class, Mapping super / parent / dimension / metaclass, Kind unit / measure / occurrence, Role measure / value instance);

Model Application domains: upper ontology (Behavior, Flow, etc.) encoded in meta model and specialization levels for domains contexts. Declarative abstractions ontology for application design: discovery, alignment and matching for services renderings and integrations.

Hierarchy: render / process layers (Relation, i.e.) as Resource, Statement, Role, etc. (upper layers) contexts (i.e.: reify Relation as Resource, Statement, Role, Kind layer contexts).

Context (reified / instances) kinds (topics): Resource content types. Resource (monad): representation, HATEOAS, dialogs. Dataflows (order / domain / range kinds).

HATEOAS: Form / Flow. Operations / Dataflow Representation / State IO (CRUD) prototypes / templates. Prompts (values / operations). Dialog. Gestures. Context: navigation state (i.e.: pick operation value prompt shows value type Form). DDD DOM.

**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).

**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.