Table of Contents

[Introduction 2](#__RefHeading__5014_1911391798)

[Metamodel 7](#__RefHeading__5016_1911391798)

[Layers 11](#__RefHeading__5018_1911391798)

[Aggregation 13](#__RefHeading__5020_1911391798)

[Augmentation 13](#__RefHeading__5022_1911391798)

[Functors 15](#__RefHeading__5024_1911391798)

[Messages 16](#__RefHeading__5026_1911391798)

[Signatures 18](#__RefHeading__5028_1911391798)

[Dataflow / Activation 19](#__RefHeading__5030_1911391798)

[Services 22](#__RefHeading__5032_1911391798)

[Encoding 23](#__RefHeading__5034_1911391798)

[RDF Quads 26](#__RefHeading__5036_1911391798)

[Semiotic Encoding 31](#__RefHeading__5038_1911391798)

[Normalization 32](#__RefHeading__5040_1911391798)

[Parsing 32](#__RefHeading__5042_1911391798)

[Protocol 37](#__RefHeading__5044_1911391798)

[Addressing 37](#__RefHeading__5046_1911391798)

[Low level 39](#__RefHeading__5048_1911391798)

[Hypermedia activation 42](#__RefHeading__5050_1911391798)

[Facets / Levels 43](#__RefHeading__5052_1911391798)

[Model Source Level (Backend) 44](#__RefHeading__5054_1911391798)

[Model Session Level 44](#__RefHeading__5056_1911391798)

[Model Interaction Level 44](#__RefHeading__5058_1911391798)

[Features APIs 44](#__RefHeading__5060_1911391798)

[Features 47](#__RefHeading__5062_1911391798)

[Domains schema 47](#__RefHeading__5064_1911391798)

[Ontology matching 47](#__RefHeading__5066_1911391798)

[Augmentations 51](#__RefHeading__5068_1911391798)

[Deployment Use Case: Goals Application 51](#__RefHeading__5070_1911391798)

# Introduction

Distributed Integration and Consistency for Semantic Knowledge Enhanced Applications.

**Description**

Distributed systems / micro services access to shared data. Shared data consistency / inference. Ontology matching. Integration (EAI / ESB). Introduction of new features / products integrating over existing (linked) data with Semantic capabilities and enhancements.

Distributed Knowledge Base. Functional Syndicated Application Integration Framework. Plug existing backends (applications / datasources / services) via Connector(s) in an EAI / ESB fashion. Provide semantic augmentation of learned applications metadata (data / schema / behavior).

**Purpose**

Objectives: Develop Protocol (APIs) to facilitate Enterprise Application Integration (EAI) by means of Semantic technologies and Machine Learning. Ontology matching driven data, schema, behavior inference / aggregation / matching. Reasoning and learning over different consolidated backends alignments.

Distributed P2P (Blockchain) approach of data synchronization between peers for ease of deployment patterns election and datasources integration (APIs, microservices, etc.).  
  
Data alignment:  
  
Determine if two instances (example: records) of two different backends or services refer to the same entity (Customers : John D. / Employees : John Doe).  
  
Schema alignment:  
  
Determine, for example, meaning and equivalences between diverse (aggregated / composite) schemas (equivalent classes, equivalent attributes, equivalent roles).  
  
Behavior alignment:  
  
Determine meaning and equivalences between (aggregated / composite) behavior contexts and behavior contexts invocations / interactions (Appointment / Interview, anAppointment / anInterview. Behavior flows aggregated from backends / services learning).  
  
Layered models semantic infrastructure for integration of heterogeneous backends (meta models).

Alignments Augmentations:  
  
Activation: type inference : classification (determine class / metaclass / roles for entity attributes and values).  
  
Activation infer attributes / relations : clustering (from multiple occurrences of same entity in diverse data sources).  
  
Aggregation: infer roles in contexts: regression (Person class in Employment interaction : Developer role).  
  
Integration of addressable resources. Reactive I/O (sync back ends). Content type driven semantic augmentation / annotations.  
  
Integrated view. Navigate contexts, data, interactions. APIs. Dimensional views annotations (analysis / mining).  
  
Augmentation of distributed resources. Annotations (Semantic / ML). API for resource / schema / interactions exploration / protocol for message based API "dialogs" execution. HAL (Hypertext Application Language), OData (REST) like interfaces.  
  
Example: Google Drive / Google Knowledge Graph APIs Augmented with ML / Semantic intelligence tailored for specific domains / application kinds.

**Features**

Graph encoding of data / schema / behavior. Dimensional / Grammar annotations. MetaGraph: augmentation / transforms (Messages). Features.

Parallel distributed graphs models augmentation / transforms synchronization (Messages). Event sourcing (distributed inferences). P2P / DIDs.

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

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.

**Description**

**Analysis (mission)**

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 for applications interoperation

**Declarative Model / Domain driven**

**Design (vision)**

Domain Business Modelling. Integration. Syndication. General purpose business domains upper ontologies for ad-hoc application building overs existing domains. Domains: Use cases “problem spaces" domain translation / exchange / integration. Hypermedia Activation and Addressing. Link extended content types resources elements / parts with other resources addressed elements in a dimensional fashion.

Domains: Use cases domains abstractions (problem "spaces" / ontologies) enabling domain translation / exchanges / integration.

Semantic components:

BI / EAI smart dashboards / reports / workflows / process / activity / indicators inference / prediction / execution. Abstract upper ontology application models. QA, polls, learning, profiles, guided task wizards / editors. Goal. Purpose. Forms. Templates. Model context to fulfill (roles / rels).

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

Solution

Integration by Augmentation.

Integration by Extension.

Declarative Application Design.

Domain Business Modelling. Integration. Syndication. General purpose business domains upper ontologies for ad-hoc application building overs existing domains.

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

Integration by Augmention: sources / back ends. Model I/O materialized in source (plugged) application / services backends. Framework inferences augment original (source) applications and services.

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

Declarative Application Design.

Features / Approach:

Data / Schema / Behavior Abstraction:

Source inputs of Connector(s) (plugged backends, applications, datasources) and data comming from declaratively stated Model interactions (Message IO) is rendered in a layered Model of Statement(s), each one representing: Input, Data (instance: Statement, class: Entity), Schema (instance: Kind / Role, class: Class) and Behavior (instance: Flow, class: Behavior) layers.

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.

Several types of Model(s) exists: Facets, each one preserving this layered structure. Model Facets have corresponding Layers and those layers are populated by corresponding Data, Schema, Behavior conforming Ontology Levels for each Facet. Facets abstract Model(s) inputs regarding this aspects: Source (Functional) Data, Semiotic and Dimensional Model Facets.

Facets are also populated in what are called Ontology Levels, which are Facet data, schema, behavior statements aggregated from feedback from the data, schema and behavior corresponding instance layers of the Facet Models themselves again into the input layer thus allowing for further describe upper ontology abstractions. These upper abstraction may be grouped into: Backend / Source (Data : plain inputs), Grammar / Session / Context (Schema : schema layer feedback inputs) and Interaction (Behavior : behavior layer feedback inputs).

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.

Protocol (deployment):

Functional "Dialog" Augmentation.

Semantics Protocol (Dataflow Message).

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.

**Domains alignment / matching**

**Sources augmentation / data virtualization**

**Implementation**

Deployment:

Container: Forms / Flow DCI / MVC Facade APIs. Resource (REST) Monad and Context Functors (reified Message resources). Protocol (Connector / Client Adapters).

Low level Resource API.

DOM / OGM rendering / activation APIs.

Model Context DOM members (layers): order (prev, next, parents, children) in axis. Metaclass, class, instance, occurrence (roles / rels in contexts). Functors / DOM events API: request reified Message resource in "possible" (domain, mapping, range) context. Bindings (subscriptions).

Trust. Consistency. Event sourcing. Inferencing (of distributed state). Reconciliation. Certify Entity / Subject Identity. Class / instance alignment (matching).

Distributed systems / micro services access to shared data. Shared distributed data consistency / inference. Ontology matching. Integration (EAI / ESB). Introduction of new features / products integrating over existing (linked) data with Semantic capabilities and semantic enhancements.

Distributed Knowledge Base. Functional Syndicated Application Integration Framework. Plug existing backends (applications / datasources / services) via Connector(s) in an EAI / ESB fashion. Provide semantic augmentation of learned applications metadata (data / schema / behavior).

Primitives / meta / upper resources. Grammars: patterns / signatures. Context Kinds (Message / Augmentation)

Kinds: hierarchies. Semiotic / Sets DM / RM encoding.

Layers. Streams. Events. Augmentation.

Resource Monad (REST). Addressable Interface. DOM (roles, rels, members, etc.) Functors.

Form / Flow Facades

Message Functor. LHS: domain: source / RHS: range: dest. Augmentation / Event pattern declaration. Mapping encoding. To do.

Model Context hierarchy (DOM). Layers CSPOs classes DOM: type / context, occurrence, attribute, value, prev, next, parent, children, etc. members. Functors.

Message: Augmentation (event). declaration.

Core: primitive / meta / upper resources. Reified layers contexts. Order (dataflow) encoding.

Core: Augmentation. Aggregation (pragmatics, roles). Alignment (semantics, rels). Activation (levels: schema, grammar, syntax of roles / rels).

Core Contexts. Layers. Augmentations.

Domains (Context DOM instances).

Dataflow: Events I/O. Message augmentations. Connector / Client / Adapter.

Message Mapping Augmentation perform. XML / XSL over RDF (cons) lists. DOM members resolution (functors) / embedding.

Aggregation / Matching (Model / Entity, Behavior / Measure). OntResource embedding (contextual functor).

Events I/O "fans in/out" model layers performing Augmentation Activation.

Graph key / value (properties) encoding. Map Reduce event flow tasks.

# Metamodel

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.

Update:

Distributed Integration and Consistency for Knowledge Semantic Interoperability.

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). 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:

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.

## Layers

What my attempts where about in the beginning was to match different URIs or, for example, database identifiers which refer to the same entity (in different databases / ontologies, for example) to perform some kind of "ontology matching".  
  
Then I've tried to develop a mechanism for using RDF Quads for encoding an object graph (and a layers class hierarchy) using Contexts to denote the class of an instance, Subjects to denote class instances and attributes (members) and values: Predicates / Objects.

Then I've realized that some basic type inference could be performed with, for example, aggregating Subjects with the same predicates (Subject Kinds). Idem for Predicates, Objects and Contexts. I've also realized that plain "facts" statements could be aggregated in the previously mentioned class hierarchy to abstract further, from plain data, instance / class layers of what I call data / schema / behavior layers. Higher layers (i.e.: Behavior) "aggregate" lower layers.

Quads are "reified" as Resource(s). Also, Resource is a functional wrapper reactive and event driven of an URI. And an URI could be implemented with whatever backend which could produce or consume events (databases, services, etc.). Resource layers hierarchy (Context) is to be implemented by an actor / role type object pattern according the hierarchy layer level it corresponds (and declaratively stated in a Model of Meta Resources).  
  
Layers shape is as follow:  
Resource : Functional URI wrapper.

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

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

Input Layer: (CSPO layer):

(Transaction, someOne, buys, someProduct);

Statement (data layer instance):

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

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

Entity (data layer class):

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

(Entity, Statement, Occurrence, Attribute);

(someTransaction, transactionStatement, someOne, buys);

Role / Kind (schema layer instance):

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

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

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

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

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

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

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

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

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

(Model Impl, Buy, someBuy, Person); Interaction / Meta Model.  
  
This "aggregations" are part of what I call "Augmentation(s)": Aggregation, Alignment and Activation are ones of those, which are functional transforms described declaratively in an object graph Meta Model. The act of applying an Augmentation implies one source Message Resource (context layer), one matching Template Resource (input signature) an Augmentation (Interaction functor) a Transform Resource (output signature) and a resulting (set of) Message Resource(s) materialized as further layers instances / Messages to be “parsed” by further corresponding Augmentations of matching Template signatures (dataflow).

## Aggregation

## Augmentation

Activation (type inference): classification (determine class / metaclass / roles for entity attributes and values).

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

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

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

Augmentation of distributed resources. Annotations (Semantic / ML). API for resource / schema / interactions exploration / protocol for message based API "dialogs" execution. HAL (Hypertext Application Language), OData (REST) like interfaces.

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.

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

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

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.

## Functors

Functors functional declaration:

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

(((O, S), P), C): Referring Context. Augmentations functors signature. Traversal performs functor augmentations "backwards" traversal direction concatenating type, role, interactions transforms incrementally.

Encoding (example): recursive CSPO IDs: (metaclass, class, instance, occurrence) IDs. URLs: domain/CID:SID:PID:OID. Graph URLs / rels traversal.

Addressing: URLs encode complete (possible) state flows: reified model state URLs (faceted browse / CRUD).

REST HATEOAS: Link rel (account): deposit, whitdraw, etc. Flow Behavior "referrer" rel.

Resource entity 'whitdrawal': context interaction. Actions 'possible'. Behavior Flow "referrer" rel.

ROC Resource: Resource set resolution (operations over subjects).

Reactive: PUT registers callbacks IRIs (subscribe to updates). Streams: Functors (hypermedia events / subscriptions dataflows).

Serialization / Links: JSON, XML (XPath, XLink, XPointer, XQuery, XForm). Encoding (rel, location, href, text). Framing / GraphQL.

State graph locators: relative lists identifiers encoding of all HATEOAS browsing attributes of a given state: logs.

As we have shown, JSON-LD itself is not a complete technology stack, it needs ontologies to express domain semantics. In future work we would like to investigate how a lightweight ontology to support a wide range of application domains could be modeled. Furthermore, we would like to explore various ideas to create smarter service clients.

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.

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.

Model / Domain levels of common model / domain monads, functors: model / domain abstract augmentations / behaviors. Declared in model contexts messages / augmentation instances.

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.

Adapter (Connector / Client):

Synchronization: Functional. Monads (source / dest: domain / range). Functors (APIs: templates / event drivers for function composition / translation). Inverse functions: backend IO protocols / formats. Adapter endpoint resolution: activates on backends protocols / formats / data.

Adapter (Connector / Client): Model Encoding. Container (reactive message / event driven) APIs. Model / Container APIs interactions.

Adapter / Model "statements" IO abstraction (Forms / Flows: Message events attribute / values).

Message (events): bidirectional CRUD streams (Adapter "template" methods, Model Message declarations):

Adapter: Context layer (semiotic interpreter).

onCreate;

onRetrieve;

onUpdate;

onDelete;

## Messages

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).

* Meta Model:
* URI;
* Resource (URI\*);
* Role (Model CSPO hierarchies) : Resource;
* Statement (Resource, Resource, Resource, Resource) : Resource;
* Kind (Statement\*) : Resource;
* Class (Kind\*) : Resource;
* Context (Class\*) : Resource;
* Hierarchy: class (Object / Value) as superclass Context.
* Object: class (extension);
* Context: super class (intention);
* (Kind, Statement, Role, Resource); Data (Resource Kind).
* (Class, Kind, Statement, Role); Schema (Role Class)
* (Context, Class, Kind, Statement); Interaction (Statement Context).
* State Facet / Layer / Level / Augmentation / Model Resource Mappings.
* Meta Resource / Meta Model:
* Meta Resource / Model: encode Model, URIs / Layers / Contexts / Facets / Levels / Resources hierarchies. Mappings.
* Meta Resource / Model: Encode Message, Template, Augmentation(s), Transforms and Mappings (Dataflow).
* Meta Model: Data, Session, Interaction Levels (Message, Template, Transform, Augmentation statements). Mappings.
* Model Context / Layers, Facets, Ontology levels, Meta Resources / Models mappings / reification. APIs. Levels example: Behavior / Interaction (Action, Gesture..., Flow). Upper ontologies: Action, Gesture etc. classes.
* Contexts / Layers / Levels / Facets Meta Resources / Models classes / instances hiers (ontology matching / data, schema, behavior alignments). Members: URIs, Resource, Context, CSPO, Meta Resource / Model APIs.
* Meta Resources are used by a Model Meta Model for describing models.
* Augmentation. Aggregation Meta Model: Describe layers contexts compositions. Alignment Meta Model: Describe augmented attributes (by kinds clustering). Activation Metamodel: Describe Kinds / Roles activation (by attributes aggregations).
* Encoding. Addressing (contents, signatures, contexts). Events publish / subscribe. Dynamic subscriptions / bindings. Subscription, reactive Meta Resource(s). Message flow mechanism: from Model to base layers.
* Encoding: Layers relations: Metaclass / Class / Instance. Subject / Occurrence / Role / Attributes / Values. Mappings declarations: Specifications, Forms, Flows.
* Subject (Resource) / Context (Statement) / Occurrence (CSPO instance) / Role (Kind) / Attribute / Value.
* Metaclass (Occurrence) / Class (Context) / Instance (Attributes / 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.
* 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.
* Ontology matching (Data, Schema, Behavior alignments):

## 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:

Messages: Events IO / Persistence: Saga Activation / Passivation populating Node local Quad store / persisting peers via DIDs ([ont.io](http://ont.io/)) semantic (resolvable / discoverable) identifiers.

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

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

Persistence: (activation / passivation): IDs / Meta Model / Facets from Interaction Model events (Messages) from Node IO. Interaction Model: Main Model(s) Message IO.

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

Saga Activation / Passivation populating Node local Quad store / persisting peers via DIDs ([ont.io](http://ont.io/)) semantic (resolvable / discoverable) identifiers.

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

Messages: Saga Passivation. Model layers data routed by Mappings as event Message into (Interaction) Meta Model. Message inputs: Models. Mappings. Populate.

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

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

Messages: Saga Passivation. Model layers data routed by Mappings as event Message into (Interaction) Meta Model. Message inputs: Models. Mappings. Populate.

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

## Dataflow / Activation

**Dataflow:**

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.

Functors: Model / Domain Augmentations (Forms / Flows). MDM. Provenance. Versions (time, dimensional / semiotic / functional axes: location, language, events, types, roles, behaviors, other rels).

Flows: Addressable interactions (Model signatures reactive bindings).

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

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).

Messages: CRUD / Domain  Invocation semantics. Flow grammars / verbs. Dialog. Prompts. Inputs are aligned into Message and are applied to Mapping Template and rendered by Mapping Transform (class extension for Augmentation class intention).

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 cuads). 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).

Query Resources by role in context.

Query Resources by attributes / values.

Query Resources by identity / type.

Streams. Subject Kind: Subjects stream. Object Kind: Objects stream. Predicate / Context Kind: Flow Signature. Stream (filter SO kinds).

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).

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.

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).

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).

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.

## Services

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.

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).

Layers (session, dialog, etc.). Node, Peer, Client, Connector, etc. Reactive / Event Driven. REST HATEOAS.

Protocol (deployment):

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

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).

**Connectors**

Backends

Tryton

GNU Health

Apache Metamodel

JBoss Teiid

JBoss Drools / JBPM

OData

R2RQ / R2RML

SPARQL

ISO 15926 / ISO Topic Maps

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

Declarative hypermedia: REST / HAL / HATEOAS

Declarative hypermedia: SOAP / WSDL

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

# 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. Embed operable meta data in CSPO contextual IDs. Vector space model like. Polynomial coefficients / factors. Adjacency matrix / tensor.

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).

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

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?).

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.

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

(Marriage (Role (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))));

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.

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: 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.

Functor applied to context: Aggregation.

Functor applied to subject: Alignment.

Functor applied to predicate: Activation.

Functor applied to object: members traversal.

Deployable entity: Node. Publish / Subscribe signatures (interface). Augmentation / Mappings Interaction Model (Runtime). Models, Facets, Services, etc. ToDo.

Grammars / Levels / Discovery (Model Forms / Flows Specifications / Protocols): Definitions: Quads, contexts, Kinds, Grammar / upper ontology as level / aggregation relationship. From data to dialog gestures. Augmentations aggregation, alignment, activation.

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:

Protocols / Services / Clients: Context interaction sessions (state flows).

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.

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.

Alignment Message: Resource -> Statements (attributes, values).

Activation Message: Statement -> Kind, Class.

Aggregation Message: Statement -> Statement (next layer).

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

Messages metamodel: functor declarations partially defined over metamodels resource (T) defining transforms into (R) over appplication (flatMap) over / into (S). Messages inferred / aligned, activated, aggregated according base message transofrms resources. Messages inferred from models / layers. TBD.

R : Model Context hierarchy.

Functors <T, R> -> Resource<R>;

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

Processor which acts upon Resource events. Materialize results. Specify declaratively augmentations by means of messages.

Upper onto / domain aggregated messages.

Event bus: P2P deployment.

Messages: Monadic applicables over Resource (flatMap). Matches Augmentation Forms / Flows.

Base HTTP / Browse (REST) Messages. Custom Messages.

Service URIs (reactive clients / connectors):

Base core service 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.

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.

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 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).

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.

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).

## Semiotic Encoding

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.

## Normalization

## Parsing

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).

**5.: Models**

**5.2.: Meta Resources**

Meta Model Context hierarchy classes:

Classes: Layers monads. Class hierarchy. Inputs resolves from wrapper containers to next layer occurrences (map forward), occurrences contexts collects matching result graph (reduce backwards).

ID<ID> : Reified matching URIs

Transform<ID> : Range

Mapping<Transform>

Template<Mapping> : Domain

Augmentation<Template>

Resource<Augmentation>

Role<Resource> : CSPO Role

Statement<Role> : CSPO Quad

Model<Statement> : Set of Statements

Meta Resource / Model: encode Model, URIs / Layers / Contexts / Facets / Levels / Resources hierarchies. Mappings.

Meta Resource / Model: Encode Message, Template, Augmentation(s), Transforms and Mappings (Dataflow).

Meta Model: Data, Session, Interaction Levels (Message, Template, Transform, Augmentation statements). Mappings.

Model Context / Layers, Facets, Ontology levels, Meta Resources / Models mappings / reification. APIs. Levels example: Behavior / Interaction (Action, Gesture..., Flow). Upper ontologies: Action, Gesture etc. classes.

Contexts / Layers / Levels / Facets Meta Resources / Models classes / instances hiers (ontology matching / data, schema, behavior alignments). Members: URIs, Resource, Context, CSPO, Meta Resource / Model APIs.

Functional API: Message IO. Mappings.

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.

**5.3.: Meta Model**

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

Monads / Functors examples (order: comparables / upper / dimensional ontologies):

Kind<SubjectClass<Resource>, Set<PredicateClass<Resource>>>;

Example: Subject / Predicate(s), others (quads prev / next relation).

Metaclass<ObjectClass<Resource>, Set<OccurrenceClass<Resource>>>;

Example: Object / Statement(s), others (layers parent / child occurrences relation).

Class<ObjectClass<Resource>, Set<ContextClass<Resource>>>;

Example: Resource / Kind(s), others (class definition relations: extension / intension, layers parent occurrences prev relation).

Reactive Uniform Component API: Context reactive instances: network addressable / operable (pub / sub streams). Wrapper API.

Addressing: Encoding. Network URLs, Semantic URNs. Naming, Index, Registry operations.

Dataflow Routes: Context Signatures. Forms. Bus. Addressing dispatch resolution.

Message: IDs Statement. Specifications (CRUD).

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

Augmentation Mappings: Flows (Wrapper API). Exchange.

Materialize Specification: Transform (Wrapper API). Reduce.

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).

ContextClass<OccurrenceClass> : Layer (IDs). Attributes, Values.

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

Layers:

Resource?

(ID, ID, ID, ID); Message (encode reified verb?)

(Transform, ID, ID, ID);

(Mapping, Transform, ID, ID);

(Template, Mapping, Transform, ID);

(Augmentation, Template, Mapping, Transform);

(Resource, Augmentation, Template, Mapping);

(Role, Resource, Augmentation, Template);

(Statement, Role, Resource, Augmentation);

(Model, Statement, Role, Resource); Resource Occurrence in Model (Kind? Kind Role type, Kind hierarchies.)

Functional layers?

Resources (reactive entity: quad / ID):

Layers Context types: reified Resource quads instances (Meta Model Kinds). Resource quad wrapping: signatures bindings.

Context layers instances: Meta Model Kinds hierarchy. Resource quad wrapping: signatures bindings.

Meta Model Kinds hierarchy (super / sub Resource class / kinds rels: super: ctx kind, sub: obj kind). Reified model entities: layers super types,

Uniform Reactive Resource Quads Wrapper API (Resource / IDs / Message / Layers): Metaclass (P) / Class (C) / Instance (O) / Occurrence (S) CSPO Resource roles / rels members, monads / transforms. Events domain / range.

Reactive component: Resource, pub / sub (endpoints APIs) for wrapping signature bindings (layers). Events domain / range.

Aggregations: Subject / Attributes, Attributes / Object, Subject / Object (Kinds). API for Functional layers interaction / composition.

Enable uniform treatment of Resources for layers aggregation / augmentation, etc.

Message / Specifications (Mappings Forms / Flows). Encoding (Specification, Form, Flow) of Mapping Transforms.

Transform. Compare: common upper types.

# Protocol

## Addressing

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

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

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

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.

Model (meta / upper resources) / DOM (domain instances) Augmentation Functors:

Aggregation: Create. Context occurrences. Materialise into next layer(s) streams. Context occurrences statement(s) (children: next layers). To do.

Alignment: Update / Retrieve. Augment attributes / values. Materialise into next context augmented statement streams (next context occurrence attributes / values, layer siblings). To do.

Activation: type / role in contexts. Parent (occurrence Context) occurrence Kind for Context object (parents: previous layers). To do.

Activation: type / role in contexts. Parent layer (occurrence context layer) CSPOs Kind(s). Materialise into dataflow Message(s) streams: (CK, SK, PK, OK). (parents: previous layers). To do.

Example:

Entity in (Model, Statement, Role, Resource).

Kind in (Entity, Model, Statement, Role).

Aggregation / Alignment Messages from primitives / meta / upper / domains resources Activation.

CSPO: Context (intension), Object (extension). SortedSet comparison hierarchies (encode order in semiotic DM / RM, octal comparison values).

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

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.

URIs: Contexts (data), Types (reified schema), Service (behavior). Class, instance, occurrence links, paths, pointers (annotations / transforms / embeddings / calculated / descriptions).

Persistence: distributed event queue (saga) of distributed addresses (occurrences metaclass, class, instance resolution). Composite occurrences (links / rels / roles).

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.

Resource: Model, graph (encoding). Dataflow: order encoding. Sort dimensional measures (SortedSet?).

Encoding: Semiotic levels metaclass, class, instance, occurrence (in context in role: metaclass?).

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

## Low level

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:

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

TBD:

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.

Meta Model (Interaction Layer Augmentation Aggregated Model declarations: facets, levels, layer).

Meta Model Interaction Layer (Augmentation: aggregated Source, Grammar, Pragma Levels Mappings) Mappings render Data, Schema, Behavior Resources for Functional, Dimensional, Semantic Meta Model Facets layers.

Entity Kind aggregation (Statements) procedure example. Encode into Quads. Alignment and Activation Quads encoding.

Context / Resource type hierarchy design pattern: plain class hierarchy,  parameterized class on Resource(s) / URIs, monads, metaclass, others. Actor / context / role (Statement CSPO position / Meta Resource). Reified Model types. DOM. DOM, Actor / Role / Context, OGM APIs.

Augmentation: transform algorithm (basic operation).

Encoding: Model (Resource).

Model: RDF Backend.

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).

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).

Augmentation: Context Resource Monads / 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).

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.

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.

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.

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

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch,  
event bus routes. URIs / IDs mappings. Resource set specification resolution. MetaGraph resolves concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics via MetaGraph driven transforms (data / schema / behavior augmentation: dialogs).

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

Models have “Facets” which renders the different ways Model data / schema / behavior could be regarded and used for different purposes, from application development to Business Intelligence and Ontology Matching.

Facets are models implemented the same way other models are with Model Resource Contexts and layers and from the same data. Each Facet implements its own Resource URI wrapper (same URIs, ontology matching, provenance of aligned URIs, Facet pivoting). Then, each Facet has its own Model Context Resource class hierarchy having Augmentation / Dataflow functors as Model Resource(s) does.

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.

Models have “Facets” which renders the different ways Model data / schema / behavior could be regarded and used for different purposes, from application development to Business Intelligence and Ontology Matching.

Facets are models implemented the same way other models are with Model Resource Contexts and layers and from the same data. Each Facet implements its own Resource URI wrapper (same URIs, ontology matching, provenance of aligned URIs, Facet pivoting). Then, each Facet has its own Model Context Resource class hierarchy having Augmentation / Dataflow functors as Model Resource(s) does.

Functors:

Functors: Meta Model declarations / Context classes / instance by hierarchies: declarative implementations of monadic functors (Levels: Augmentation / Domain Flows). 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.

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.

### Model Source Level (Backend)

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 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).

## Features APIs

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.

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.

Resource monad of Model(s) layers Contexts: Transform, Kind, Entity, Behavior, etc. Model(s) Context hierarchy classes (model) instances (domain). Contexts, Occurrences, Attributes, Values : CSPO Functors (eval in Resource contexts). Functor declarations (model augmentations / domain dataflows encoding).

Monads / Functors examples (order: comparables / upper / dimensional ontologies):

Kind<SubjectClass<Resource>, Set<PredicateClass<Resource>>>, others. Example: Subject / Predicate(s). (quads prev / next relation).

Metaclass<ObjectClass<Resource>, Set<OccurrenceClass<Resource>>>, others. Example: Object / Statement(s). (layers parent / child occurrences relation).

Class<ObjectClass<Resource>, Set<ContextClass<Resource>>>, others. Example: Resource / Kind(s) (class definition relations: extension / intension, layers parent occurrences prev relation).

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 (augment / extend): RDBMS (R2QL / R2ML) / Rules / BPM, KIE / Teiid / Metamodel / Olingo / Solid / ERP / CMS / Feeds (Hypermedia Resources / Apache Stanbol) / Browser / Messaging (JMS overlay, Apache Camel / ActiveMQ: ServiceMix Connectors / Clients) / ML, Big Data (Apache Spark).

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

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

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

Models: Augmentation, Kind (Metamodel), Entity, Class, Behavior (Functional), Dimensional, Semiotic (Matching / Alignment / Upper).

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.

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

Endpoints: Model (classes) objects / resources, Domain (instances) objects / resources

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

JBoss Teiid / Apache Metamodel: Clients / Connectors / Backend / Runtime (Translators).

CMS / Wiki API / WebDAV. Apache Stanbol. DOM / REST / JAF.

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

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).

# Features

## Domains schema

Message based Augmentation Events Dataflow. Augmentation Mapping Dataflow allowing to embed dynamic state in Model entities (including Mappings Augmentations themselves).

## Ontology matching

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)

Functors:

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). TBD.

From Object (O) extension / instances to Context (C) intension / class. Matching grammar shapes. 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").

Behavior flows functor composition: Behavior, Flow, Class, Kind, Entity layers aggregation. Determine type, role, alignment augmentations. Example: type (Class Model) in context (Flow Entity) in interaction (Behavior Kind).

Navigation (TMRM, key / value) functors:

Keys.

Remote keys.

Local values.

Key / value proxies.

Value proxies.

Constraints.

Merge.

Legends (constraints).

Path languages.

Path expressions.

Encoding:

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).

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);

## 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.

# **Deployment Use Case: Goals Application**

Integration by Augmentation:

Model I/O materialized in source (plugged) application / services backends. Framework inferences augment original (source) applications and serviced.

Integration by Extension:

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

Declarative Application Design.

Semantic components.

BI / EAI smart dashboards / reports / workflows / process / activity / indicators inference / prediction / execution. Abstract upper ontology application models. QA, polls, learning, profiles, guided task wizards / editors. Goal. Purpose. Forms. Templates. Model context to fulfill (roles / rels).

Distributed systems / micro services access to shared data. Shared data consistency. Ontology matching. Integration (EAI / ESB). Introduction of new features / products integrating over existing (linked) data with Semantic capabilities and enhancements. Domains: use cases “problem "spaces" (distributed domain translation / exchanges / integration).

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.

Solution

Hypermedia Use Cases (Ontology Levels). Integration by Augmentation: Aggregation / Alignment / Activation Annotation of distributed resources. Augmented Content types driven. Encoding / Addressing (links / browse / parts / rels / roles). Microformats (embedding). Wiki like abstract interfaces / representation (indexes).

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).

Augmentation:

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

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

Alignment: Infer layer missing / deducible attributes and values.

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

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.

Reactive Event driven Model Representations.

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).

Approach

Distributed P2P (Blockchain based) approach of semantic data persistence and synchronization between peers for ease of deployment patterns election and data sources integration (client APIs, microservices, etc.). Layered models semantic infrastructure for integration of heterogeneous backends (meta models).

Source inputs of Connector(s) (plugged backends, applications, datasources) and data coming from declaratively stated Model events interactions (Message IO) is rendered in a layered Model of RDF Quad Statement(s), each one aggregating input data into a Meta Model for further functional facet abstractions into data / schema / behavior layers, dimensional layers and semiotic layers.

Integration by augmentation: sources / back ends. Model I/O materialized in source (plugged) application / services backends. Framework inferences augment original (source) applications and services.

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

Declarative Application Design.

Trust. Consistency. Event sourcing. Inferencing (of distributed state). Reconciliation.

Certify Entity / Subject Identity. 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).

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 (Hypermedia Application Language), OData (REST) like interfaces.

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.) addressable information resources inference and resolution. 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.

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.

Graph encoding of data / schema / behavior. Dimensional / Grammar annotations. MetaGraph: augmentation / transforms (Messages). Features.

Parallel distributed graphs models augmentation / transforms synchronization (Messages). Event sourcing (distributed inferences). P2P / DIDs.

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

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.

Transactions: Request IDs. CQRS. Reified interactions / gestures. Event sourcing / logs. Reified (resource) Message activation (dataflow) Forms / Flows.

Reference Model / Encoding:

Syntax: data (signs). Semantics: schema (concepts). Pragmatics: behaviors (objects CRUD / Interactions / Contexts).

Reify able roles. Agregate semiotic Activation of a statement resources roles. Example: aggregate kinds / entities. Instance (extension) / Class (intension): Object / Concept.

Order / hierarchies: reified Context / Concept roles relation (example: Concept as Context, Concept aggregations are siblings / children of first Concept).

CSPO Reference Model: (attribute / value). Reified roles (Semiotic layer). Order. Hierarchies.

Encoding: To do. Compatible with Message Augmentation Mappings Predicate comparison and Index traversal.

Index: To do. Compatible with Message Augmentation Mappings Predicate comparison. Encoding representation (cons list, graph, tree, etc.). Truth values (flows) traversal.

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).

Predicates: quad encoded comparison specification, result of iterating Mapping comparisons until Augmentation Message matches all Resource set. Augmentation comparison behavior. Message / Augmentation application specification.

Mappings: comparison iterations matching Message / Augmentation Predicates patterns truth values. Compare input Message resource set with Mappings till truth values are all OK. Perform Augmentation.

Model encoded Augmentation (Aggregation, Alignment, Activation) determined Comparison, Predicates, Mappings.

Encoding: IDs. Relative / contextual metadata. Serialization. Index.

Event: Message (specification), Augmentation (instance).

Predicate: comparison specification (axis). Message SPO.

(Message, LHS, Predicate, RHS);

Mapping: comparison result (patterns). Augmentation SPO. Template Transform rendered in Message context resources set.

Abstract Adapter / Connector / Client APIs:

Populate Model Context layer (Model Statements).

Protocol: Form / Flow Context (interaction). I/O data / schema / behavior input / output abstraction. Domain (model driven) interfaces / schema.

Inputs (data): Resources augmented / aggregated from OntResource until Dimensional layer.

Protocol (schema / behavior): Browse from Dimensional up to OntResource layers selecting, creating and matching intermediate Context(s). Form / Flow interaction use case roles.

Aggregation / augmentation: feedback from interactions (levels) CUD interactions.

Adapter Meta Resources (upper). Translate inputs. Render outputs from Model protocol into backends adapter specific protocol APIs.

Languaje levels: Use, mention. Pronouns.

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

Adapter Model: metaclass, class, instance, occurrence. Context (interpreter), Sign, Concept, Object.

Connector I/O (gestures):

Client I/O (dialog):

Model layers:

OntResource: Resolves reified aligned / matched aggregated Resources.

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

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

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

Model:

(OntResource, OntResource, OntResource, OntResource);

OntResource represents aggregated / matched different identifiers / URIs referring to the same subject.

(Predicate, OntResource, OntResource, OntResource);

For a Predicate occurrence, attributes / values.

(Message, Predicate, OntResource, OntResource);

For a Message Predicate occurrence, possible attributes.

(Context, Message, Predicate, OntResource);

Occurrence (object) for a Context (interpreter) Message (sign) Predicate (concept). Adapter: Context layer (semiotic interpreter).

(Transform, Context, Message, Predicate);

(Mapping, Transform, Context, Message);

(Template, Mapping, Transform, Context);

(Augmentation, Template, Mapping, Transform);

(Resource, Augmentation, Template, Mapping);

(Role, Resource, Augmentation, Template);

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

(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.

Ontology Matching: Equivalent propositions that for different subjects have the same dimensional layer equivalent Measure(s) Value(s): attributes / values.

Example: Application (protocol) shows aggregated Measures Dimensions, select Unit / Value and assert / browse Measures.

Pick (matched / new) Behavior corresponding to Measures Values. Select available / new Flow.

Pick Flow Class and assign Kind (DCI Role). Assign / create Entity (model alignment / assignation).

Follow up in occurrences hierarchy: CRUD / CUD available / possible. Perform Augmentations.

Products And Services Community Exchange Network:

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

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

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

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

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

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

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

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

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

Ontology:

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

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

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

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

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

Backend: Nodes / Protocol.