# Encoding

Augmentation: Layers (occurrences, aggregations). Quads.

Upper Ontology Roles Aligned Object Models. Templates.

Roles: Metaclass, Class, Instance: Resource Occurrence, Resource, Kind, Context.

**Meta Model:**

Object Model:

Labeled Property Graph.

Serialization (Aligned Quads):

Occurrence: (Context, Object, Concept, Value);

Occurring: (Object, Context, Concept, Value);

(Context : Concept type / label, Object, Concept, Value) DOM Property Graph.

(Object, Context : Concept type / label, Concept, Value) DOM Property Graph.

Layers APIs:

Connector Bus. Templates, Forms Meta Model Functional Dataflow. Functional DOM.

**Encoding:**

Object Model:

Labeled Property Graph.

Serialization (Aligned Quads):

(URN, Context, Kind, Resource);

Layer APIs:

Addressing, Matching, Persistence. Graph Model Functional Dataflow. Functional DOM.

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.

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.

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) events as Messages / Transforms. Reified Model entity types / roles (CSPO, Kinds, Layers, etc.). Addressable interactions: DIDs.

## Encodings

The whole idea revolves encoding 'roles’ (Resource types) in an aggregated dimensional layered (data, schema, behavior) scheme via the use of 'kinds’ abstraction.

**Kinds abstraction:**

sets / type inference. Given an statement SPO, for example the Subject, its 'subject kind’ is the aggregation of all Predicate / Value pairs occurring in statements having that Subject. Subjects having a common set of this pairs are regarded as having the same 'kind’. Subset / superset relations between those pairs determine super / sub 'kind’ relations.

If we only regard of Predicate occurrences for a common Subject we can have a very basic class / type inference also with a predicates subset / superset (class / superclass) relation. The same holds for 'predicate kinds’ and 'object kinds’. A class could be, for example, Person and kinds (metaclass) of occurrences of this class could be: Employee, Father, etc.

Event (definition): measure value change in dimension for unit. Kind: stream of state occurrences (bus: roles functional predicate).

**Activation:**

Resource I/O events stream (quads). Materialize new / inferred knowledge, emit known facts regarding event.

### Aggregations:

Object occurrences aggregates into Predicate occurrences which aggregate into Subject occurrences and then in Context occurrences.

Entity: Plain RDF URLs input.

Subject example: (S, SPO, P, O);

Kind: Entity occurrences aggregation.

Subject Kind Example: (Kind, Entity, P, O);

Class:

(Class, Kind, SPO, Entity);

Flow:

(Flow, Class, Kind, Entity);

Behavior:

(Behavior, Flow, Class, Kind);

Upper:

(Behavior, Flow, Kind, Class);

## Contexts

**Resource wrapper:**

Aggregated internal layered (data, schema, behavior) quad statement sets. Behaves as a data, schema or behavior layer via activation composition (Resource I/O).

Interacts (I/O activation) via upper ontology layer aggregated form facade statements: (Behavior, Class, Entity, Resource).

Resource IDs / namespace handler.

Index, Naming, Registry facades.

Addressing / annotation (augmentation) resolution of external resources. JAF.

Augmentation: Aggregation, Alignments, Activation.

Reactive streams: events, locators, filters, transforms, queries, aggregation, getters / setters, joins, etc.

Functional Forms: code / data stated as resources. System resources. Bound functions / transforms.

Layered aggregated contexts stack:

(Context (Application (Domain (Data))))

Functional Forms: Browse layered aggregated context (render services / applications).

Layers integration: dimensional resources 'overlay’ interacting via activation of upper ontology layer aggregated form statements.

Data contexts: data, schema, behavior aggregated I/O from plain RDFized (Resource IDs) inputs.

Domain contexts: data, schema, behavior aggregated I/O from Data contexts aggregated upper ontology.

Application contexts: data, schema, behavior aggregated I/O from Domain contexts aggregated upper ontology.

## Resource IDs

URNResource: CSPO map: (this : cls/ids (ctxs : cls/ids (occur : cls/ids( role : cls/ids)))).

The idea is to achieve a (numbering) identification scheme which allows to encode and identify RDF statements CSPOs URLs (and the URLs referring to the statements itselves) in a manner which:

1. Allows to ‘embedd’ meaning in an algorithmically 'operable’ way.
2. Enforces preservation of 'validity’ between identifiers (no non-valid identifiers could be forged)

In base to the abstract layer (semiotic) statement form:

(Context, Concept, Sign, Object);

The idea is that (in theory) using a positional ternary numbering system with a cyclic order relation (a > b > c < a) CSPO IDs could be validated against:

C > S < P < O

For any given statement IDs arrangement.

Primitives:

self > this > that < the

Alignments:

X is Y of Z in W;

C(S, P) : O;

ML Embeddings.

### ID Assignment

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

S : Concept, P : Sign, O : Object.

(the, self, this, that);

Context::nextID(URL / ID, leftLastID, currPosLastID, rightLastID);

C : nextID < S : nextID > P : nextID > O : nextID;

Meta Model: Verb Occurrence / Occuring action / state / passion (Contexts).

## Functional Forms

Functional code / data (codat / runat) serialization format / language expressed in terms of Resource statements.

TID: Statement Context ID.

VID: Statement Subject (Occurrence) ID.

TID:VID: Context / Occurrence attribute / value sets (recursive forms).

Form:

(TID:VID (TID:VID (TID:VID (TID:VID)))) : TID:VID;

(Behavior (Flow (Class (Kind )))) : Dimensional abstract Resource (Entity);

Assertion / query language. Activation.

Algorithm resolves over Behavior, Schema, Data attributes / values.

Dataflow activations: candidates for resolution (signatures / injection).

Specific system forms (augmentation bound functions).

DOM / LINQ like APIs.

ML Embeddings.

### Functional Activation

(Context, Occurrence, Attribute, Value);

(Entity, Statement, Attribute, Value);

(Kind, Entity, Statement, Attribute);

(Class, Kind, Entity, Statement);

(Flow, Class, Kind, Entity);

(Behavior, Flow, Class, Kind);

Form:

(TID:VID (TID:VID (TID:VID (TID:VID)))) : TID:VID;

(Behavior (Flow (Class (Kind )))) : Dimensional abstract Resource (Entity);

OGM: TMDM / TMRM. FCA, Mappings.

**Mappings:**

Resolve Flow from Behavior, Class from Flow, Kind from Class, Entity from Kind via mapping matching:

(O -> P); (P -> S)

From lower layers to upper layers.

## Features

DCI: Data, Context, Interaction.

(Context, Interaction, Data:role, Data:state);

Data (event), Information (flow), Knowledge (rule: context, role, state flow).

Type inference: Entity, Statement, Kind, Class, Flow, Behavior. Functional Form syntax, upper ontology.

Dataflow graphs: dynamic 'routes’ (Resource stream observer / observable) 'signatures’ (activation matching: resolution / injection). Aggregation (layers), composition (contexts), discovery.

Integration:

Integration of federated sources of knowledge and invokable behaviors (applications, services, backends, etc.) in a transparent and uniform manner enabling ease of distribution and composition into new aggregated assets.

Data Consistency in Microservices Architecture:

Microservices: domains common classes / interfaces (behaviors) : DOM / OGM Layer. Declarative description of original business (legacy addressing / activation).

API Gateway. Composite service (aggregator). Database (Context) per service.

Command Query Responsibility Segregation (CQRS). Events sourcing / views. Saga: compensating requests. Logs: aggregate standarized log format for each service instance.

Distributed tracing: external request unique ID. Sub requests contextual 'external' IDs (graph). Record logs.

Externalize configuration. Command pattern.

Service discovery: registry. Locate by interface / schema signatures (capabilities).

Java. Functional Resource API. DOM / OGM. DDD / JAF.

RDF / OWL Backend.

Declarative Resource REST API Gateways (I/O: aggregation, forms activation, etc.).

Reactive Framework. Container. Addresses (Resource IDs mappings).

Messaging. Dataflow. Routing (Resource IDs / Address mapping). Layers. Injection (Forms activation).

Augmentation: Aggregation, Alignment, Activation. (ID / Class, Attributes, Roles Alignments).

ML Declarative (functional) services API (general shapes / models). Naming, Registry, Index. Serving (activation shapes).

ML Context Index Service: dot notation graph encoding. Templates, grammars, activation. Facets. Class / ID Resolution.

ML Context Registry Service: attributes / links in contexts resolution.

ML Context Naming Service: dictionary: key / value (recursive) store. Synsets. Senses (contexts / from SPOs occurrences). Embeddings. Role in context resolution.

Domains (Upper ontologies) DOM, OGM Context metadata for: CMS, B2C, B2B, ERP, CRM, SCM, ESB, EAI, BI (business domains).

Endpoints Rendering: Declarative resource described / driven APIs. Microservices.

Clients Rendering: Declarative activation DCI / MVC contexts / flows. Portlets.

Adapters: JBoss Teiid. Apache Metamodel. OData. SoLiD. HAL / HATEOAS. GraphQL.

XML / XSL / XLink / XPath / XPointer.

JavaScript.

APIs:

Factory methods. Builders. Visitors. Enum for case classes. Functional architecture. Patterns.

Node: Connectors, I/O, services. Address resolution (activation events / messages) of Node domain Contexts (Resources). Backends.

Application, data, reference models. Graphs. Sets, Groups, Categories models.

**Templates: Resource specifications**

Template: map. Role Kinds match (types). Activation / aggregation. Node Resources traversal. Factory / Builder.

**Forms: Transform specifications**

Form: reduce. Role Kinds apply (flatMap). Activation / aggregation. Node resources materialization. Factory / Builder.

**Ontology Matching**

Specification : Resource (Context);

Node: Connectors. Resources feed. Custom URLs for different services / backends I/O.

Resource layer: Resource instance for each one CSPO roles in quad statement input, Specification for the rest of CSPOs in Resource layer statements. Example for Subject: (Specification, Subject, Specification, Specification).

Aggregation into Statement layer by Resource and roles Specification matching. Resource are instances from resource layer (with Specification in roles). SPOs in Statement occurrence role w./ attrs & values.

Refine Specification "learning" until matching Context Subjects, Predicates and Objects unambiguously from inputs.

**Functional mapping of role kinds to resources**

Smart contracts.

**Resource IDs: IDs Graph. Addresses**

Resources DID (W3C Distributed Identifiers) implementation. DID Documents: API contracts / HAL.

Unique Resource ID: URL.

Unique (distributed) sequence ID.

Forge Resource hierarchies URLs.

Forge Class / Kind URLs.

Layers Aggregation rules (from Behavior to Resource):

(C: prevOcc, S: nextCtx / prevPred, P: nextSubj / prevObj, O: nextPred);

Implement Specification (Template / Form) for previous rules.

IDs Graph (quads):

ResourceID: (ResourceClsID, ResourceInstID, StatementID, RoleKindID);

StatementID: (StatemenClsID, StatementInstID, ResourceID, RoleKindID);

RoleKindID: (RoleKindClsID, RoleKindInstID, StatementID, ResourceID);

Address:

did:ont:clsID:instID/clsID:instID/clsID:instID

Signatures (Specification): TBD.

Discovery: TBD.

Composite query expressions building Resource IDs Graph statements.

Aggregation (“counting”) of sequence IDs. Tree. Primitives (the, self, this, that). Next, prev, curr ID functions.

Encode orderings in class relationships. (via role kind).

Encode orderings in instances relationships (via role kind).

Encode orderings in class / instance relationships (via role kind).

Addressing / annotations of resources (XSL, XLink, XPath, XPointer). Activation of content types.

**Upper ontology: Initial Resource Activation layer**

(Dimension, Unit, Measure, Value);  
(Context, Concept, Sign, Object);  
(Context, Occurrence, Attribute, Value);

Root (IDs Graph hierarchies) Resources. Initial activations upon upper ontology.

APIs: Activation transforms / functional bindings. Use dimensional, grammar, syntax representations.

**Domain / Upper ontologies**

ERP Ontology, ISO, TMDM, TMRM, SKOS. Datasources: GKG, Wikimedia. Specifications: Shapes.

**Computation over graph structures (dataflow): layers class / occurrence orderings**

Ordering can be defined by the properties of their relations. Order defined by relations / occurrences and Role Kind subset / superset relation.

Graph: Node set, Edge set, Attributes, Global attribute. G : (u, V, E).

DGI: “every single node of the network is ‘mindful of the global structural properties of the graph’”. Representations.

Compute updated edge attributes.

Aggregate edge attributes per node.

Compute updated node attributes.

Aggregate edge attributes locally.

Aggregate node attributes locally.

Compute updated global attribute.

Three update functions (mappings).

Three aggregate functions (reductions).

Combinatorial generalization in graph networks.

**Encoding: Services Models, Backends, ML (Graphs, Embeddings, Autoencoders + Context)**

Resource occurrences: mode (tense / person / primitives: the, this, that, self, etc) "declination" (modal) metadata, inferred from contexts / aggregation / role kinds (role kinds role kinds specifications: from contexts / statements, negation, complement, equivalence, etc.). Ordering by properties of relations. Parameterized control flows / interpretations. Octal order encoding for flow control.

Property graphs.”The DOT Language “. N3 / Turtle rules.

Property dot graph encoding. Grammar: rules / productions. Specification (functional) DOM like notation / serialization / selectors (monads in specification runtime / components events / callbacks codat / runat).

Resource services (Naming, Registry, Index). Connectors: declarative contexts. Resource URL scheme for handling backends / services kinds I/O.

Distributed consistency

Higher level constructs (declarative language, APIs)

Use cases (layers "patterns": DOM / DCI / CDI / JAF)

Bindings with declarative metamodel description (layers "tags", templates, activation)

Application Language (component blueprints: layers). "Templates" from declarative metamodel descriptions

Applications: domains upper ontologies. Distributed consistency. Data / schema / behavior metamodel instances inferences / rules.

URNResource: CSPO map: (this : cls/ids (ctxs : cls/ids (occur : cls/ids( role : cls/ids)))).

**Augmentation:**

Object Model:

Sets CSPO Model.

Serialization (Aligned Quads):

Sets CSPO Model Statements encoding.

Layer APIs:

Aggregation, Alignment, Activation. Sets Augmentations Functional DOM:

* Data (Data): key / value. Column: (price: 100). Data Aggregation Augmentation.
* Information (Schema): Record (keys / values relation): (price: 100, brand: ACME). Schema Alignment Augmentation.
* Knowledge (Behavior): Records (columns values relationship). Price variation behavior example: ((price: 100, brand: ACME, date: today, priceVariation: 0), (price: 110, brand: ACME, date: yesterday, priceVariation: 10)). Activation Augmentation: materialize relationships / facts.

**DDD REST HATEOAS DOM**

Object Model:

ID: Object Occurrence.

Object (ID, Type, Member\*); Node.

Type : Object;

Member : Object; Arc (Property Graph).

Serialization (Aligned Quads):

(Object, ID, Type, Member\*);

Layer APIs:

Naming, Registry, Index. DOM Functional Dataflow.

Messages: Dynamic Object Model Functional Monads bound (kinds signatures subscriptions) Functions. Contexts (Data, Schema, Behavior) browse traversal / transform. Resource aggregates Messages.

(Resource, Transform, Mapping, Statement);

Connector Bus API. Messages.

Layers Dataflow Layout. Messages.

Templates: Activation. Messages.

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).

Basic Message aggregation (Context Mapping): shift right mapped applied statement resources. Mapped resource context > instance (occurrence) of next layer message reified resource context.

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.

**Functional DOM**

Static:

Layers Object Models Domain Wrapped.

Layers Objects Models Domain Wrappers.

Layers Object Models Domain Functions.

metaclass::class::instance::context:::occurence::role::Resource;

Resource<Subject>

Subject<Resource>

To do: APIs.

Dynamic:

Layers Model Instances Domain Wrapped. Resources.

Layers Model Instances Domain Wrappers. Kinds.

Layers Model Instances Domain Functions. Contexts.

Transforms: DOM Resource Roles traversal functions, i.e.: Resource Roles (Kind) functional getters.

(Peter : Person (Employee : Kind (empPeter : Manager)));

**Transactions Log API: DLT / Events Sourcing**

Message: Transform Log. To do.

ResourceURNs: Uniform identifiers across occurrences. DID URN. Endpoint. ResourceURN Statements: uniform functional metadata (contextual type / name, relations / aggregated occurrences). IDs Encodings.

Templates / Monads / Sets Interfaces. Graph layout. Traversal (Quads Monads). Set Membership Function: Interfaces CSPOs Types Matching Signatures.

Interface types differentiate in their CSPOs return value types (CSPOs type signatures: sets membership function).

Input / Canonical: Match Interfaces / Signatures: (Context, Occurrence, Attribute, Value);

Attribute / Value Roles in matching interface context. Order / hierarchy encoding: functions (sorted wrapped functional collections: wrappers set comparators / aggregation in axis).

Interface Quads Matching determine Sets (intersections) membership.

Hashing: IResourceURN, IOccurrence, IKind, IResource. Nested recursive URNs aggregations. Aggregations / Order / Mappings / Traversal APIs.

ResourceURN Occurrence, Kind, Resource Bindings Augmentation Service APIs:

* Inputs: Aggregate SPO into CSPO: Aggregates Contexts Type / Table / Class Kinds. Aggregate PK Cols, Cols : Occurrence, Val : Resources.
* Inputs (Rel / Graph): (Type / Table / Class, PK : Resource, Col : Occurrence, Val : Resource).
* Inputs (Rel / Graph) FKs: Val : Resource equivalent PKs.
* Augmentations / Transforms:
* Data matching. Resource equivalence: identity / comparisons / order transforms.
* Schema matching. Predicates equivalence. Domain / Range types. Order: data flow contexts.
* Behavior matching. Domain / Range values applied functional predicates identity. Order: data flow interactions.
* Transforms: DOM Resource Roles traversal functions, i.e.: Resource Roles functional getters.
* metaclass::class::instance::context:::occurence::role::Resource;

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

Classes: Layers monads Contexts class hierarchy. Inputs resolve from wrapper containers to next layer occurrences (map forward), occurrences contexts collect matching result graph (reduce backwards). Map / Reduce: Graph key / value / properties encoding, signatures reactive bindings).

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

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.

Semiotic:

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

(Context, Occurrence, Attribute, Value);

(Context, Sign, Concept, Object);

Metaclass, class, instance, occurrence.

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

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

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

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

Dimensional:

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

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

Assert knowledge: 1h -> 60min;

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

Assert: 1mt -> 100cm; etc.

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

Order / comparisons: tree representation ordered by Context Role class hierarchy, instances hierarchies and aggregation hierarchies. Resources order (IDs). Statements order (Statement IDs). Comparison criteria (choose relevant IDs). ToDo. SortedSet hierarchies.

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

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

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

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

Ontology Matching:

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

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

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

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

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

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

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

Messaging metamodel:

(Message, Resource, LHS, RHS);

(Interaction, Message, Resource, LHS);

(Role, Interaction, Message, Resource);

(Context, Role, Interaction, Message);

(Dataflow, Context, Role, Interaction);

Encoding:

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

Kinds, Signatures. Contents. Contextual metadata. Lattices. Roles. Sets (bitstring cuads). Definitions (elements). Operations. Rules. Categories. Groups. SortedSet hierarchies (3 digit octal set membership values).

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

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

Quad encoding: Context relative IDs.

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

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

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

Data order: Resource Kind hierarchies.

Schema order: Role Class hierarchies.

Interaction order: Statement Context hierarchies.

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

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).

**Facets:**

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: Type 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).

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.

**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.

Message : Contexts

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 (data flow). 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.

**Protocol**

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.

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.

**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.

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.

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.

**Augmentations**

Augmentations:

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.

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

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

Encode common upper Semiotic / Dimensional Model: Reference Model.

Encode Kind / Context hierarchies.

Encode Augmentation(s) as Resource descriptions.

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

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

Object occurrence of Predicate.

Sets. Quads. SortedSet.

Metaclass / Class / Instance.

Class / Instance ID pairs:

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

Semiotic encoding:

(Context, Sign, Concept, Object);

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

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

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

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

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

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

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

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

Augmentation: Aggregation (Context template).

Augmentation: Alignment (Attribute, Value template).

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

**Topics**

* 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):