* MDM. Provenance. Versions (time, dimensional / semiotic / functional axes: location, language, events, types, roles, behaviors, other rels). ISO. Rules. BPM. KIE.
* Message (Form / Flow) / Resource: Meta Model parent classes? Specification / Protocol.
* Signatures: Mappings Context Kinds. Possible Flows, actual Augmentation.
* ContextClass<OccurrenceClass> : Layer (IDs). Attributes, Values.
* Events: Monads (IDs 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?).
* 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.
* 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
* Layers:
* (ID, ID, ID, ID);
* (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);
* Facets.
* Key / Value graph encoding. Map Reduce. Flows: Mapping declarations / assertions (possible flows).
* 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.
* Kinds, Signatures. Contents. Contextual metadata. Lattices. [Roles.Sets](http://roles.sets) (bitstring cuads). Definitions (elements). Operations. Rules. Categories. Groups.
* 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.
* (Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.
* (Measure, Value, Previous, Distance);
* (Unit, Measure, Value, Previous);
* (Dimension, Unit, Measure, Value);
* (Concept, Dimension, Unit, Measure);
* (Resource, Concept, Dimension, Unit);
* (Statement, Resource, Concept, Dimension);
* Populate / align / annotate models with dimensional data. Model input: statements (model resources). Model specification: augment, sort statements. Model specification: specialization of base model layers. Resolve resolution statements order.
* Dimensional input set model specificatíon (from Statement layer, ordered SPOs: order criteria, comparisons. Kinds / class / occurrence / instance order criteria?).
* Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements layers).
* Value -> distance(prev, next); ordering;
* Assert knowledge: 1h -> 60min;
* dom-lun-mar-mie-jue-vie-sab (orders);
* 1mt -> 100cm; etc.
* Comparison / order: Alignments (prev, curr, next asserted knowledge). Next hour, location, city, country, next distance at next time at current speed. Event sourcing / tracking: married -> marriage occurred.
* Sort: cause / effect, temporal, etc. Messages align, functional map, fold, etc. Primitives. Encode layered statements ordering. Complement / supplement concepts definitions.
* 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.
* Events declarative definition. State change of value in axis in measure of context.
* Messaging metamodel:
* (Message, Resource, LHS, RHS);
* (Interaction, Message, Resource, LHS);
* (Role, Interaction, Message, Resource);
* (Context, Role, Interaction, Message);
* (Dataflow, Context, Role, Interaction);
* Basic hypermedia browse / CRUD (HTTP verbs) bound Message functors compatible for all Resources (REST).
* [Resource.flatMap](http://resource.flatmap)(Message::apply) : Observable<Resource> (stream). Composable functions.
* 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.
* OntResource is the class responsible for aggregating different URIs referring the same entities (Ontology Matching).
* Resource : Functional (Monad) OntResource wrapper.
* (Context : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);
* (Statement, Occurrence, Attribute, Value);
* (Entity, Statement, Occurrence, Attribute);
* (Role, Entity, Statement, Occurrence);
* (Class, Role, Entity, Statement);
* (Flow, Class, Role, Entity);
* (Behavior, Flow, Class, Role);
* 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.
* Flows: Addressable interactions (Model 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).
* 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: (Message applicable signatures : resolution result: Transform).
* Query Resources by role in context.
* Query Resources by attributes / values.
* Query Resources by identity / type.
* Context Kind: Functional stream of Context Statements (Occurrences).
* Subject Kind: Functional stream of Subject Statements (Occurrences).
* Predicate Kind: Functional stream of Predicate Statements (Occurrences).
* Object Kind: Functional stream of Object Statements (Occurrences).
* Messages: Dataflow Template matches signatures (Session level, enrichs Message with Model / Dialog prompts / contents 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.
* 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).
* 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.
* Cons lists. Binary Trees. Huffman / Prefix codes. RDF List serialization. Meta Resources / Models declarative statements Encoding, Addressing.
* Kinds, Signatures. Contents. Contextual metadata. Lattices. Roles. Sets (bitstring cuads). Definitions (elements). Operations. Rules. Categories. Groups.
* Statement: (ID (ID (ID (ID, Nil))));
* (C (S (P (O, Nil))));
* 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.
* 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.
* Streams. Subject Kind: Subjects stream. Object Kind: Objects stream. Predicate / Context Kind: Flow Signature. Stream (filter SO kinds).
* Messages: Saga Activation. Interaction Model (Meta Model). Aggregated (Interaction) Meta Model interactions (performed / inferred / possible) emitted as Model event Messages (Saga pattern). Mappings.
* Dataflow: Order, Flows (Mappings, hierarchies).
* State order (in context class hierarchies axes), comparison relations, iterations, flow, events, causal relations, units, enums, equivalence, etc.
* Data order: Resource Kind hierarchies.
* Schema order: Role Class hierarchies.
* Interaction order: Statement Context hierarchies.
* IDs: Addressing / Encoding. Semantic (signature, contents, context) resolvable / discoverable identifiers.
* Meta Resource / Models / Messages: IDs / Encoding / Addressing formats. Ontology matching and Template / Augmentation / Transform enrichment (alignments), transforms (functors), materialization (model updates) via Mappings (events) and Meta Resource / Model Encoded Resource declarations (enrich / align, transform, updates algorithms: Encodings).
* 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.
* 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.
* 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.
* Context / Object class by intension / extension. Transform matches Context signature, filters by Object(s) extension. Resource(s) specification.
* Reified Kind(s).
* Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs [mappings.Resource](http://mappings.resource) set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.
* 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.
* [Resource.flatMap](http://resource.flatmap)(messageInst::apply) : Resource.
* Dataflow: Messages hierarchy. Aggregate contexts from coarse to fine grained  transforms (raiseSal -> setAttr).
* data <-> schema <-> behavior.
* 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.
* 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).
* Base HTTP / Browse (REST) Messages. Custom Messages.
* Addressing: URIs, DIDs URLs. Adddress, content type, representation (URI APIs). Browse / CRUD (DAV).
* Resource<T : URI> monadic hierarchy. Basic hypermedia browse / CRUD (HTTP verbs) bound Message functors compatible for all Resources (REST).
* [Resource.flatMap](http://resource.flatmap)(Message::apply) : Observable<Resource> (stream). Composable functions.
* Basic Message application (Context Mapping): shift right mapped applied statement resources. Mapped resource context> instance of mesage reified resource context.
* Transforms (Message templates):
* Resource: Statement
* Message: SubjectKind
* Transform: Resource
* Resource: Statement
* Message: ClassLayer
* Transform: Statement (class)
* Resource: Employee
* Message: Position
* Transform: Manager
* Dimensional / Grammar models.
* 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.
* Encodings
* aX\*4 + bY\*3 + cZ\*2 = dW
* d, a, b, c: classes (CSPO)
* WXYZ: instances (CSPO)
* powers: CSPO role
* terms: CSPO resources
* Z(obj) is Y(pred) for X(subj) in W(ctx).
* instance, class, metaclass, ocurrence terms.
* Wildcards, primitives, variables, placeholders. resolution (Discovery, DIDs).
* Templates (grammars / message parsing in template context: mapping transform)
* Subjects: attr / val
* Occurrence: ctx / role
* Behavior: order / compare.
* Proof of work
* MetaGraph model: map URIs -> IDs
* Satisfy dW. Sync resolution (recurse terms contexts)
* FCA. Resource attributes. Tensor, adjacency matrix, tree.
* Encode behavior in statements / graph:
* Comparisons, order. Sort. Order (kinds hierarchy?) Pattern matching, iteration, jumps. Discovery: routes / signatures, next event in bus / graph.
* 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.
* 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.
* Value -> distance(prev, next); ordering;
* Assert knowledge: 1h -> 60min, lun-mar-mie-jue-vie, 1mt -> 100cm.
* Comparison / order: Alignments (prevv, curr, next). Next hour, location, city, country, next distance at next time at current speed. Event sourcing / tracking: married -> marriage occurred.
* Sort: cause / effect, temporal, etc. Messages align, functional map, fold, etc. Primitives.
* Encode layered statements ordering. Complement / suplement concepts definitions.
* Ontology matching (table, pk, col, val example). Helper upper models for models linking / alignment.
* Event sourcing (“offline” sync). API
* Graph linking / alignment / sinchronization by entailments from event sourcing over inferred state. Reconciliation.
* 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.
* Meta Model default Augmentations:
* Aggregation classification. Registry svc. Alignment regression. Index svc. Activation clustering. Naming svc.
* Context Kind Signatures.
* Datasources / Backends / Services. URIs. Signatures: dataflow (Context Kinds). CKs Attribute / Value (SK / PK) determines domain / range I/O of a Resource / URIs.
* Ontology matching (Backend / Interaction Model).
* 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):
* Kind: Basic type inference. Applied over layers CSPO during Activation Augmentation. An Occurr
* Augmentation: Event. Signature. Declarations / Occurrences. Domain, Input / Mapping, Transform / Range, Output. Specifications: Forms, Flows. Embeddings (dataflow / specification references), Meta Model / domains encoding (levels).
* Augmentation. Aggregation Meta Model: Describe layers contexts compositions (regression). Alignment Meta Model: Describe augmented attributes (by kinds clustering). Activation Metamodel: Describe Kinds / Roles activation (by attributes aggregations classification).
* Semiotic Facet:
* (Attributes, Occurrence, Attribute, Value);
* (Object, Attributes, Occurrence, Attribute);
* (Concept, Object, Attributes, Occurrence);
* (Sign, Concept, Object, Attributes);
* (Context, Sign, Concept, Object);
* (Interaction, Context, Sign, Concept);
* (Model, Interaction, Context, Sign);
* Data / Information / Knowledge:
* (Properties, Occurrence, Attribute, Value); Data (Properties: distance / facts).
* (Value, Properties, Occurrence, Attribute); Info (Properties distance between Occurrence / previous and Occurrence / next).
* (Measure, Value, Properies, Occurrence); Knowledge.
* (Unit, Measure, Value, Properties);
* (Dimension, Unit, Measure, Value);
* (Concept, Dimension, Unit, Measure);
* (Model, Concept, Dimension, Unit);
* Example:
* (Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.
* Order layers statements. Hierarchies (contexts / kinds). Parent / child relationships (steps). Order type relationships: husband: single / marriage / married.
* (Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.
* (Measure, Value, Previous, Distance);
* (Unit, Measure, Value, Previous);
* (Dimension, Unit, Measure, Value);
* (Concept, Dimension, Unit, Measure);
* (Resource, Concept, Dimension, Unit);
* (Statement, Resource, Concept, Dimension);
* Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements layers).
* Value -> distance(prev, next); ordering;
* Assert knowledge: 1h -> 60min;
* dom-lun-mar-mie-jue-vie-sab (orders);
* 1mt -> 100cm; etc.
* Comparison / order: Alignments (prev, curr, next asserted knowledge). Next hour, location, city, country, next distance at next time at current speed. Event sourcing / tracking: married -> marriage occurred.
* Sort: cause / effect, temporal, etc. Messages align, functional map, fold, etc. Primitives. Encode layered statements ordering. Complement / supplement concepts definitions.
* 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.
* 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.
* 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).
* Message. For each layer perform each Functor: (Object : aggreg, Kind : activ, Attr : align, Obj : onto).
* Augmentation:
* Functors: Augmentation declaration: Meta Model definitions (Context class / instances). Message: dataflow matches Template signatures: interactions. Apply Augmentation Functors over Message contents (interactions enrich Message with Models contents: ontology matching / Levels / Facets). Materialize / emit dialog / prompts Message (enrich Message from Models / reactive IO events).
* Augmentation: Context / Functors. Message Resource(s) / Meta Resource(s) (nested / wrapped) elements determines flow Template Transform results / behaviors (CRUD, Functor invocations). Message IO performs Augmentations. Ontology levels resolution (Templates / Transforms / Augmentatiom levels: matching patterns / dialog prompts in Ontology levels).
* 12.1: Augmentation Functor(Message, Message) : Events
* Augmentations: matching Events Functors aggregate / align / activate (classify) sour
* Augmentations: matching Events Functors aggregate / align / activate (classify) sources of ontology matched data / schema / behavior enabling semantic layers interoperation.
* Aggregation: Infer input data streams data, schema, behavior class / instance context layers.
* Alignment: Infer layer missing / deducible attributes and values.
* Activation: Infer layer CSPO Kind / Roles. Basic type system.
* Aggregation (Augmentation): Apply each Context (layer) Functor on inputs (from input layer) and emits Transform, matching corresponding (next) layer. Next layer Context and SPO according functional mapping declared by Meta Resource types on augmented layer.
* Alignment (Augmentation): ToDo.
* Activation (Augmentation): ToDo.
* 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.
* Ontology Matching: Encode: order, iteration, flows, units, relations, events, enums, etc.
* Semiotic / Dimensional alignment. TBD.
* Ontology Matching. Semiotic. Dimensional. Sets. Functional Reference Model.
* Semiotic / Dimensional alignment / aggregation layers (lower resource alignment layers):
* (Context, Sign, Concept, Object);
* (Value, Distance, Prev, Next : in Units); (Measure, Value...) (Unit, Measure, Value,...); (Resource, Unit, Measure, Value); Marriage example.
* Meta Model (Meta Resources)
* 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.
* 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.
* Contents / Features (Mision / Vision). Distributed consistent Knowledge Applications.
* 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).
* 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).
* 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.
* 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.
* URIs API for annotating network retrieveable resources metadata. Content type / model driven augmentations / activations (models features / outputs). Subject attributes / values. Occurrences contexts / roles. Paths, pointers, locators. Example: annotate document URIs (parts, sections, mentions), annotate images URI (whole image description, coords: classes, individuals), annotate DB, table, row, column, value URIs, annotate / describe service / APIs URIs. Hypermedia protocol composable with other (described / annotated) APIs / resources. Example: Drive APIs.
* 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 Kind / Context 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).
* Ontology Matching. Upper ontologies. Primitives.
* Addressing / IDs / Encoding.
* Encoding: Resource ID. Encoded Resource contents (signature / occurrence). Augmentation: Resource 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).
* Encode common upper Semiotic / Dimensional Model: Reference Model.
* Encode Kind / Context hietarchies.
* 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.
* 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 (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.
* 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).
* 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.
* Encoding: Layers relations: Metaclass / Class / Instance. Subject / Occurrence / Role / Attributes / Values. Mappings declarations: Specifications, Forms, Flows.
* 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):
* The aim is being able to describe models using models themselves, maybe translating relations to Model Quad Statements.
* The same relations could be used to build a Model in which declaratively state model dataflow behavior (reaction to events). A dataflow specification could be described by the following meta resources (roles):
* Message (Subject : Data level)
* Template (Context / domain : Session level)
* Augmentation (Occurrence, declarative / service Resources: functors. Interaction level)
* Transform (Role / range: Kind transform matches. Session level). Resulting Message Attribute / Value roles populated.