* Reactive Activation: Inferences. Model Rules: N3 / Turtle / DSL. Templates: Resources, Kinds, Contexts Encoding (roles) for Functional Reasoning (Predicates: schema / values)
* XML / XSLT DTDs / XSD. RDFS / OWL OGM (Templates Encoding).
* Alignment Inferences (Functional Predicates): sameAs, greaterThan, lessThan, equals, partOf, parentOf, siblingOf, previousOf, nextOf, roles (schema / values).
* Functional Composite inferred Predicates:
* greaterThan([a.age](http://a.age), [b.age](http://b.age)) : older(a, b) : Activation. Matching, Sort.
* Predicates: Templates. Resources, Kinds, Contexts Encoding (Function predicates argument mappings). Composite from primitives / roles (Contexts).
* [business.products.premium](http://business.products.premium)
* Inference (Functions same results) Ontology Matching.
* Messages: Message semantics (Augmentation: Verbs, CRUD, Behavior) according Message structure / pattern (dialog / prompts). CQRS. Reactive: Message matching inferences (R, CUD) results.
* **Order:**
* Mappings from Resource Types hierarchy lattice.
* 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.
* Encoding: Magic numbers. Resource Content Type Hash. ID Hashing: block (DIDs) result of inferences chain (event sourcing). Encoding: addresses.
* Encoding: Model declared as Interaction Layer Augmentation(s) (matching Messages) in Interaction Model. Flows. Model: possible inferences (dataflow).
* **Message service URIs:**
* Contextual (statement / dialog) service. Inference / Augmentation Functional Predicates (mappings / classifications / clustering / regression / services) invocations.
* Example: Subject (image URI / resource : source), Predicate (detection service / index service), Object (detection / search results endpoint / placeholder : destination).
* Grammars: Predicate Kind (face / search recognition signature) from Subject (faces images / names) / Object (face classes / subjects) Kinds. Kind model layers.
* AI for Understanding Human Goals
* "In the quest to capture ... social intelligence in machines, researchers from MIT’s Computer Science and Artificial Intelligence Laboratory (CSAIL) and the Department of Brain and Cognitive Sciences created an algorithm capable of inferring goals and plans, even when those plans might fail."
* "... ability to account for mistakes could be crucial for building machines that robustly infer and act in our interests ... Otherwise, AI systems might wrongly infer that, since we failed to achieve our higher-order goals, those goals weren’t desired after all. We’ve seen what happens when algorithms feed on our reflexive and unplanned usage of social media, leading us down paths of dependency and polarization. Ideally, the algorithms of the future will recognize our mistakes, bad habits, and irrationalities and help us avoid, rather than reinforce, them."
* <https://scitechdaily.com/new-mit-social-intelligence-algorithm-helps-build-machines-that-bette>
* ("Inference" is used broadly herein to mean any rule or procedure that produces new assertions from existing assertions -- not just conventional inference engines or rules languages.)
* Furthermore, applications often need to perform custom "inferences" (or data transformations) that are not convenient to express in available (non-standard) rules languages, such as RDF data transformations that are needed when merging data from independently developed sources having different data models and vocabularies.  And merging independently developed data is the \*most\* fundamental use case of the Semantic Web.
* One possibility for addressing this need might be to embed RDF in a full-fledged programming language, so that complex inference rules can be expressed using the full power and convenience of that programming language.  Another possibility might be to provide a convenient, standard way to bind custom inference rules to functions defined in a programming language. A third possibility might be to standardize a sufficiently powerful rules language.
* Here’s a JavaScript-based language for path queries, which reduce things such as “the user’s list of friends” to three words ([user.friends.label](http://user.friends.label)) instead of a SPARQL query:
* – <https://github.com/solid/query-ldflex>
* – <https://solid.github.io/ldflex-playground/>

**To do**

* CQRS. Monads Functions (domain / range) CUD Commands, R Retrievals applicable in contexts / roles: DDD (signatures / dataflow).
* CoSQL. Duals. LinQ / DSL / Parser Combinators. Templates.
* Aggregation:
* Parse Sets Model input Statements into AST Monads. Parsed Model Execution: Parse Event Bus I/O Model Message Monads Dataflow (sync: AST Events).
* Alignment:
* Encoding / addressing Sets Model population. Versioning. Parsed Model Execution: Parse Event Bus I/O Model Message Monads Dataflow (sync: AST Events updates).
* Activation:
* Alignment Augmented DOM Model: Parse Sets Aggregation AST : DOM AST Monads. Parsed Model Execution: Parse Event Bus I/O Model Message Monads Dataflow (sync: AST Events).
* Sets Model: (hierarchy Resource root, Kinds, Contexts children). Upper Ontology (DDD).
* Contexts: CSPO Layout (Kind Roles): Functions (Interactions).
* Statement: Data
* Mapping: Context
* Transform: Interaction
* Kinds: Roles (Contexts).
* Subject: Data
* Predicate: Context
* Object: Interaction
* Resources: Values (Data).
* Subject: Data
* Kind: Context
* Resource: Interaction
* Monadic Wrapper: Category.
* Wrapped: ASTs.
* Function: Domain / Range
* Event Bus: Message Monads / Functions.
* Core Models AST Functional Dataflow Traversal:
* CQRS Models Protocol: Sets Model Monads / Functions Events encoding / addressing:
* CUD: (Context, Statement, Mapping, Interaction);
* Context::Statement::Mapping::Interaction Aggregation. (CUD).
* (R), Selector: (Context, Kind, Statement, Resource);
* Context::Kind::Statement::Resource Aggregation. (R: Selector).
* Message Model Functional Dataflow Traversal:
* Encoding / Addressing Message Model:
* (URN, Statement, Kind, Resource); Subject Position: Query / Command.
* DOM Message Model:
* (Class : Instance, Instance, Attribute : Class, Value : Instance);
* Class::Instance::Attribute::Value.
* Models Integration: Core Upper Metamodel (Encoding / Addressing). Models. URNResource: Endpoints, Reactive HATEOAS Resource Monads (Categories), Functions declarations (domain / range signatures dataflow).
* Streams in Context. Functional Reactive HATEOAS Monads / Functions. Verbs / Types reified as Resources. Alignment Augmentation.
* Identity Transform (Context): Retrieval.
* Template = Context: Create.
* Update / Delete: Versioning.
* Models ASTs Monadic domains Functions.
* Functional API (Monads Functions / Wrappers (Domain / Range): Sets Object Model:
* Example: Resource<Subject>, Subject<Resource>;
* CoSQL. Duals. Meijer.
* API Functions (domain / range: individual subjects / streams in context):
* getResource / getResources
* getKind / getKinds
* getContext / getContexts (Statement, Mapping, Transform)
* getContext / getContexts (CSPO)
* getSubject / getSubjects
* getAttribute / getAttributes
* getValue / getValues
* getMetaclass / getMetaclasses
* getClass / getClasses
* getInstance / getInstances
* getContext / getContexts
* getRole / getRoles
* getOccurrence / getOccurrences
* Merge TOCs.
* Merge Contents.
* Content / Topics:
* Concepts, Design, Architecture.
* Add bibliography / tools use cases / components. Bookmarks, Lectures. Notes: Scrapbook.
* Concepts:
* Data, Information, Knowledge.
* Data, Schema, Behavior.
* Models: Layers Message IO Dataflow Bus.
* Models: Sets Contexts, Kinds, Resources Layer.
* Events Sourcing / Models Bus IO.
* Input Message Augmentations:
* Aggregation: Populate Sets.
* Alignment: Addressing / Encoding / Matching.
* Functional Activation Dataflow API:
* Resource Monad
* Kind Monad
* Context Monad
* Message Monad
* Event Monad
* API: Dataflow:
* API: Command. CQRS (CUD, R): Context (Mapping Contexts).
* API: Event: Command / Message (Context, Template: D, Mapping:  C, Transform: I);
* Dataflow: Event dispatching. Event Message / Command Context augmentation.
* Activation: Topics reacts to Events according API. Context, Kind, Resource Chain of Responsibility. Performs CUD/R and a response stream relevant to the operation performed.
* API: onEvent(Event) : Event. Order / Comparisons / Workflows.
* Activation: Statements.
* Template Data Roles (Kinds) selectors / predicates Matching Statements. Data.
* Activation: Mappings.
* Statements Matching Mappings. Schema / Context.
* Activation: Transforms.
* Mappings Matching Transforms. Behavior / Interaction.
* API: Core Model Bus Topics: Contexts, Kinds Resources.
* API: Core Model Transforms / Mappings Functions.
* API: Dynamic Model (instances) Bus Topics: Resources, Kinds, Contexts.
* API: Dynamic Model (instances) Transforms / Mappings Functions.
* Models: Addressing / Encoding / Matching Layer.
* ResourceURNs Occurrences (Subjects):
* (ResourceURN, Resource, Kind, Context);
* (ResourceURN, Resource, Context, Kind);
* (ResourceURN, Context, Kind, Resource);
* (ResourceURN, Context, Resource, Kind);
* (ResourceURN, Kind, Context, Resource);
* (ResourceURN, Kind, Resource, Context);
* Addressing: Model Traversal: MapReduce
* ResourceURNs Contexts, Resources, Occurrences IDs Addressing / Encoding:
* ResourceURN : (ContextResourceURN, SubjectResourceURN, OccurrenceResourceURN);
* Matching: ResourceComparator(s).
* Models: DOM OGM / DCI / DDD / CDI: Restful objects Layer.
* RDF DOM OGM / DCI / DDD / CDI: Sesame Elmo.
* Resources / Kinds / Contexts: DOM / DCI / DDD Subjects, Concepts, Mixins, Behaviors.
* Functional Dataflow API. Model / Events Bus sync.
* DCI: Qi4j / Apache Zest (RDF / KeyValue / EAV EntityProvider).
* Dynamic Functional Contexts: Scala DSL Message Dataflows. Monadic Parser Combinators.
* Behavior / Factory: Roles. Parameterize (domain context) monadic functions applications.
* 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 Dataflow
* Encoding:
* Object Model:
* Serialization (Aligned Quads):
* (URN, Context, Kind, Resource);
* Layer APIs: Addressing, Matching, Persistence.
* Augmentation:
* Object Model:
* Sets CSPO Model.
* Serialization (Aligned Quads):
* Sets CSPO Model Statements encoding.
* Layer APIs: Aggregation, Alignment, Activation. Sets Functional Dataflow Augmentations.
* 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.

**Scrapbook**

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.

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);

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

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

Contexts:

* 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.

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

Flows: Addressable interactions (Model signatures reactive bindings).

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

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.

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

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

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:

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).

Backend:

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

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

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

Functor applied to context: Aggregation.

Functor applied to subject: Alignment.

Functor applied to predicate: Activation.

Functor applied to object: members traversal.

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

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

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

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

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

Deployment / Use Cases:

Purpose driven hypermedia activation:

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

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

Browser referring context (Work, Peter, Employee).

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

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

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

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

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

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

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

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

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

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

Activation Message: Statement -> Kind, Class.

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

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

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

R : Model Context hierarchy.

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

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

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

Upper onto / domain aggregated messages.

Event bus: P2P deployment.

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

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

Service URIs (reactive clients / connectors):

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

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

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

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

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

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

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

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

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

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

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:

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).

* 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):

**Notes**

**Mision / Vision**

**Objectives**

**Description**

**Use Cases**

**Problems**

**Solution**

**Approach**

**Features**

**Reactive event driven**

**Augmentation Dataflow**

**Ontology Matching**

**RDF Introduction**

**RDF Objects mapping**

**Models**

**Layers**

**Interaction Layer**

**Data**

**Schema**

**Behavior**

**Meta Model**

**IDs: Encoding**

**Facets / Levels**

**Dataflow**

**Resource APIs: Monad(OntResource)**

**Model functional APIs: factories**

**Reactive streams (kinds, etc.)**

**Message: Monad(Resource)**

**Model Functional APIs : Message builder**

**Functional APIs: reactive streams (Augmentation / dynamic routes)**

**Persistence**

**Dataflow**

**Model Interaction Layer matches Messages**

**Interaction Layer Augmentation dataflow**

**Meta Resources (Template, Transform) / Transform dataflow embeddings**

**IDs: Encoding**

**Messaging: Addressing / Discovery**

**Augmentation: Functor(Message, Message) : Event**

**Dataflow Model Events declaration**

**Meta Resources: Interaction IO (Messages Meta Resources bindings / embeddings / prompts / mappings)**

**Aggregation**

**Alignment**

**Activation**

**Service**

**Connector**

**Ontology Matching**

**Index**

**Registry**

**Naming**

**IDs: Encoding**

**Ontology Matching**

**Augmentation Services**

**ToDo**

**Mision / Vision**

**Objectives**

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

ToDo.

**Description**

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

ToDo.

**Use Cases**

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

Solutions:

Integration by Augmentation.

Integration by Extension.

Declarative Application Design.

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

Domains: Use case. Problem "spaces" / domain translation / exchanges / integration.

Domains: Use case. Problem "spaces" / domain translation / exchanges / integration.

Semantic components:

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

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

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

Semantic components:

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

ToDo.

**Problem**

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

ToDo.

**Solution**

Integration by Augmentation.

Integration by Extension.

Declarative Application Design.

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

ToDo.

Description:

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

Problem description:

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

Use Cases (problem / solution):

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

Solutions:

Integration by Augmentation.

Integration by Extension.

Declarative Application Design.

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

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

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

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

Declarative Application Design.

Features / Approach:

Data / Schema / Behavior Abstraction:

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

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

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

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

Ontology matching (Data, Schema, Behavior alignments):

**Approach**

Develop Protocol (APIs) to facilitate Enterprise Application Integration (EAI) by means of Semantic technologies and Machine Learning. Ontology matching driven data, schema, behavior inference / aggregation / matching. Reasoning and learning over different consolidated backends alignments for applications interoperation.  
  
Distributed P2P (Blockchain DIDs) approach of data synchronization between peers for ease of deployment patterns election and datasources integration (client APIs, microservices, etc.).

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

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

Declarative Application Design.

**Features**

### Ontology matching

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

ToDo.

### Augmentation Protocol

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.

ToDo.

### Reactive / Event Driven

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

ToDo.

Features / Approach:

Data / Schema / Behavior Abstraction:

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

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

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

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

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

Protocol (deployment):

Functional "Dialog" Augmentation.

Semantics Protocol (Dataflow Message).

Applications:

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

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

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

ToDo.

Augmentation:

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

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

Alignment: Infer layer missing / deducible attributes and values.

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

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

Protocol (deployment):

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

Applications:

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

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

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

**Use Cases**

Domains: Use case. Problem "spaces" / domain translation / exchanges / integration.

Semantic components:

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

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).

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.

# RDF Introduction: Graphs, Triples, Quads

ToDo.

# RDF Quads for Object Graph Representations

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

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

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

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

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

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

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

Value as Occurrence of Attribute in Attribute Occurrence Context.

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

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

Meta Model: Layers Resource relations:

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

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

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

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

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

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

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

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

ToDo.

# Models

**Models**

Models aggregates input I/O / Connectors data into corresponding knowledge Facets (Functional, Semiotic, Dimensional).

Base Model structure / Context layers hierarchies is as follow:

OntResource (URIs).

Resource : Functional URI 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 layers in class / instance roles (except for input layer) and each upper layer aggregates functionally over the previous:

Model (Facet) Statement declaring /aggregating Model in Meta Model is of the shape:

(Model : Model Impl., Behavior, Flow, Class); Interaction / Meta Model.

Classifying (aggregating) previous layers statements as parts of the Model.

Input Layer (Resource).

Data Layer (Statement instance, Entity class).

Schema Layer (Kind / Role instance, Class class).

Behavior Layer (Flow instance, Behavior class).

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.

Models:

URI(s);

OntResource; Merged URI(s) wrapper.

Resource (OntResource CSPO / Contexts hierarchies Monad wrapper);

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

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

Augmentation : Functor.

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

Encoding: Template Message augmentation (inputs).

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

Encoding: Transform Message augmentation (outputs).

Interaction Model:

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

**Model Layers**

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

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

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

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

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

Input Layer: (CSPO layer):

(Transaction, someOne, buys, someProduct);

Statement (data layer instance):

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

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

Entity (data layer class):

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

(Entity, Statement, Occurrence, Attribute);

(someTransaction, transactionStatement, someOne, buys);

Role / Kind (schema layer instance):

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

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

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

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

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

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

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

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

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

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

One also could Augment Resource(s) in a functional manner, using reactive event driven APIs so, for example applying "Person" class to "Employee" role could shield a Resource set of people being working for someone. The ultimate goal is to be able to "plug" as much "backends" connectors as posible into distributed peers which exposes protocols / APIs for knowledge driven hypermedia applications.

**Model 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.

Functional (Model) Facet:

Resource : Functional URI 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);

(Model, Behavior, Flow, Class);

Semantic / Semiotic Facet:

Resource : Functional URI wrapper.

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

(Attributes, Occurrence, Attribute, Value);

(Object, Attributes, Occurrence, Attribute);  
(Concept, Object, Attributes, Occurrence);  
(Sign, Concept, Object, Aytributes);  
(Context, Sign, Concept, Object);  
(Interaction, Context, Sign, Concept);

(Model, Interaction, Context, Sign);

Dimensional Facet:

Resource : Functional URI wrapper.

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

(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);

**Model Ontology Levels**

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

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

ToDo.

**URIs, Resource, Contexts Functional APIs**

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.

Functional API: Message IO. Mappings.

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

Data order: Resource Kind hierarchies.

Schema order: Role Class hierarchies.

Interaction order: Statement Context hierarchies.

ToDo.

**Order:**

Mappings from Resource Types hierarchy lattice.

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.

**Meta Resources**

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. Some of them are:

URI

Resource

Context / Context

Subject / Occurrence

Predicate / Attribute

Object / Value

Statement

Model

Kind

ContextKind

SubjectKind

PredicateKind

ObjectKind

Message

Template

Augmentation

Transform

Class

Metaclass

Instance

**Meta Model**

Meta Model: encode Layers, Contexts, Kind / Roles hierarchies (subject, context, occurrence, roles, atributes, values / metaclass, class, instance relations / meta resources).

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

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

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

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

Reify Model Layers, Levels and Facets in a Meta Model with Meta Resources. Use Meta Resources class relations for describing models. Meta Resources describe components and roles of Models according a set of relations:

Subject (Resource) / Context (Statement) / Occurrence (CSPO instance) / Role (Kind) / Attribute / Value.

Metaclass (Occurrence) / Class (Context) / Instance (Attributes / Values).

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.

Meta Model:

Meta Resource class / instance patterns.

Participation: Subject in Occurrence.

Role: Participation for Subject.

Kind / Context hierarchies.

Subject, Participation, Occurrence, Roles, Atributes, Values / Metaclass, Class, Instance class / relations / meta resources.

(Participation, Role, Attribute, Value);

(Subject, Participation, Role, Attribute);

(Occurrence, Subject, Participation, Role);

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

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

Context::flatMap(ctx : Context) : Context

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.

Meta Model for Encoding / Addressing (Event routes) dataflow metadata.

Messages: Message semantics (Augmentation: Verbs, CRUD, Behavior) according Message structure / pattern (dialog / prompts). CQRS. Reactive: Message matching inferences (R, CUD) results.

### Models: 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.

#### Functional Model Facet:

(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);

(Model, Behavior, Flow, Class);

#### Semantic / Semiotic Model Facet:

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

(Attributes, Occurrence, Attribute, Value);

(Object, Attributes, Occurrence, Attribute);  
(Concept, Object, Attributes, Occurrence);  
(Sign, Concept, Object, Aytributes);  
(Context, Sign, Concept, Object);  
(Interaction, Context, Sign, Concept);

(Model, Interaction, Context, Sign);

#### Dimensional Model Facet:

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

(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);

ToDo.

### Models: Levels

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

#### Model Source Level (Backend)

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

ToDo.

#### Model Session Level

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

ToDo.

#### Model Interaction Level

Aggregate Session Level Behavior layer Statements as Model Data level Data layer input. Reify behaviors (context / interactions).

Declarative application protocol use case upper ontology levels (Action… Gesture, etc).

ToDo.

## Meta Resources

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

URI

Resource

Context / Context

Subject / Occurrence

Predicate / Attribute

Object / Value

Statement

Model

Kind

ContextKind

SubjectKind

PredicateKind

ObjectKind

Message

Template

Augmentation

Transform

Class

Metaclass

Instance

(Resource, ?, ?, ?);

(Role, Resource, ?, ?);

(Statement, Role, Resource, ?);

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

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

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

## Dataflow: Mapping

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

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

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

Message

Model (Functor)

Augmentation (Addressable Interaction)

Template (Message)

Mapping (Functor) : Inferences

Transform (Message)

Model (Functor)

Message

**Models: Quads, Contexts, Occurrences, Attributes, Values.**

Declarative means of using RDF quads to state application object models (data, schema and behavior).

**Message service URIs:**

Contextual (statement / dialog) service. Inference / Augmentation Functional Predicates (mappings / classifications / clustering / regression / services) invocations.

Example: Subject (image URI / resource : source), Predicate (detection service / index service), Object (detection / search results endpoint / placeholder : destination).

Grammars: Predicate Kind (face / search recognition signature) from Subject (faces images / names) / Object (face classes / subjects) Kinds. Kind model layers.

**Use Case**

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.

**Meta Resources**

Model declared as Interaction Layer Augmentation(s) (matching Messages) in Interaction Model. Flows. Model: possible inferences (dataflow)

Transform result adds new Context / Occurrence, merges existing Context Occurrence in Model with new Statements (prompts / embeddings). Dataflow inferences.

Augmentation: Event (Augmentation) emits new dataflow Message Transforms from Template Mappings.

Augmentation: Input Templates prompts. Dialog, Meta Resource embedded Templates / Mappings / Augmentations in input Message / output Transform.

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

Model OntResource Augmented with Event Transform aggregates new Event Mapping.

**Kinds**