**Meta Meta Model:**

DOM OntResource: Models Meta Meta Model. Reference Model. Runtime / dataflow APIs.

Parsing: AST Object URL monad layers hierarchy. Parser combinators. Monad Zippers (tree encoding). Order. Flows (streams). Parser combinators.

OntResource:

Reactive (producer / consumer) entity. Hierarchy templates determines signatures / dataflows (topics / queues).

Signatures: observer (topics): ObjectID events stream. URL events stream. Role events stream. Layers hierarchy events set (reified: Resource, Statement, Kind, Relation events). Filter. Template / abstract methods. Core OntResource API behaviors. Dataflow.

Signatures: observer (topics): ObjectID events stream. URL events stream. Role events stream. Layers hierarchy events set (reified: Resource, Statement, Kind, Relation events). Filter. Template / abstract methods. Core OntResource API behaviors. Dataflow.

OntResource abstract APIs:

Abstract: Meta Model objects occurrences (aggregated by Object URL and Roles):

OntResource::ObjectID;

OntResource::Occurrences<URL, Role> : OccurrenceID;

OntResource represents aligned Objects / Resources of which the same Object (ObjectID) occurs in different Roles and (possibly) diferent URLs. Matching and alignments by reifying layers into abstract resources / OntResources.

Meta Model(s) APIs (Augmentation, Dataflow, Dimensional) reflects reference model events reacting to and producing signature matching messages on Meta Model(s).

OntResource's Roles are the DOM (Dynamic Object Model) types of the Meta Model(s) layers Occurrences / Objects / Contexts / SPOs. FCA Lattice contexts occurrences metadata. DOM APIs (reifying hierarchy layers, abstract templates):

Role: occurrence / object in CSPO slots. Denotes resource types in positions in statements (i.e.: Kind in Relation). Role CSPO is object / occurrence in statement occurrence position, Role type (i.e.: Kind, Relation) stated as Role instances in Meta Models with corresponding Kinds for its complimentary CSPO resources.

(Instance, Class, Metaclass, Occurrence);

(Context, Occurrence, Attribute, Value);

Meta Model(s). Augmentation APIs hierarchy:

Object<URL> : OntResource;

Abstract. OntResource aligned URL / template method design pattern.

Object<URL>;

Abstract. ObjectID.

Occurrence : Object, Object<URL> : OntResource;

Abstract ObjectID, OccurrenceID. Reify aggregated hierarchies.

Role : Occurrence, Occurrence, Object<URL>;

(Resource : Role, Role, Occurrence, Object<URL>);

(Kind : Resource, Resource, Role, Occurrence);

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

(Relation : Statement, Statement, Kind, Resource);

Class hierarchy APIs:

(Instance, Class, Metaclass, Occurrence);

(Context, Occurrence, Attribute, Value);

**Meta Model(s)**

Dataflow APIs hierarchy:

Roles reify to Meta Meta Model Roles:

(Relation, Statement, Kind, Resource);

(Augmentation, Mapping / Predicate, Transform / Template, Resource);

Dimensional APIs hierarchy:

Roles reify to Meta Meta Model Roles:

(Relation, Statement, Kind, Resource);

(Dimension, Unit, Measure, Value);

Semiotic APIs hierarchy:

Roles reify to Meta Meta Model Roles:

(Relation, Statement, Kind, Resource);

(Context, Sign, Object, Value);

Domain Models. Service APIs. Templates.

Dataflow, Dimensional and Domain Models declaratively stated / reified in root Meta Meta Model Roles.

Messages. Encoding / I/O:

See 'Service Resources'.

(Class, Instance, Member, Value);

Augmentation:

Domain (Templates)

Meta Model (Activation)

Meta Meta Model (Augmentations

Meta Model (Activation)

Domain (Templates)

Encodings:

(a (b (c (d, nil) : First / Rest binary tree.

(K: (K: inst, V: cls), V: (K: mcls, V: occur))

Object / Occurrence: Reify layers Object / Occurrence keys / values. Reference Model along with OntResource.

Message I/O (see Service Resources):

(Class, Inst, Attr, Val);

Templates: I/O Dataflow (aggregated messages) declarative bindings. Aggregated message facade (Augmentation, render / apply layer entity schema / data). Forms / Flows.

Forms / Flows: HATEOAS HAL. MVC. REST. Meta Models DCI based protocol.

HATEOAS: Forms / Flows Operations / Dataflow Representation / State IO (CRUD) prototypes / templates. Dialog. Prompts. Gestures. Context: navigation state (i.e.: pick operation value prompt shows value type Forms). DDD DOM.

Models Sets Reification (Populate DOMs / FCA):

OntResource

Object

Occurrence

Role : CSPO hiers Sets.

Resource : Role Set member.

Statement : Set members Role aggregation.

Kind: Statement Resources aggregations. Roles intersection sets.

Relation: Kind Statements aggregations. Transform: Kind Resources related to themselves (ID), then Relations to other Resource via Dataflow Kinds domain / range relationship (ordered).

Relation: aggregated aligned entities. Views (transforms). Kind members occurring in Statement Resource(s). Functors / Monads:

Relationship<A : Relation>::flatMap(F : Function<A : Relation, B : Relation>) : Relationship<B : Relation>;

Entity<A : Relationship>::flatMap(F : Function<A : Relationship, B : Relationship>) : Entity<B : Relationship>;

Function: declarative dataflow transform.

Dataflow Kinds domain / range: Grammar. Reify Kinds as CSPO and assert Statement. Aggregate further Kinds (until primitives).

Valid Statement (Grammar / Relation): domain / range, CSPOs backing assertions apply.

I/O: Parse / aggregate input Statements into corresponding Roles / Resources. Aggregate / match Kinds. Relations: render / activate. Resolve output Statements.

Built in Relation(s): ID, equals, inverseOf, parent, child, previous, next, etc. (upper ontology / meta model). Composites: Monad Zippers.

Meta Meta Mode (TBD)l:

DOM: OntResource, Object, Occurrence, Role. Express Meta Model levels metaclass, superclass, class, instance, occurrence relationships.

(Superclass, Class, Occurrence, Instance);

Metaclass: Role.

Occurrence: Role instance.

**Monadic Functors / Transforms:**

Parsed from / rendered to (materialized) to Meta Meta Model DOM Relations Monad Statements. Inheritance from Relation, Entity, Relationship, Flow.

Domain Models Entities / Relationships / Flows: Model parsed / rendered Relation materialized domain functors / transforms. Monadic layer for functional operation of underlying DOM Model Relations.

Model Relation:

(Relation, Statement, Kind, Resource);

ToDo: Resource selector API. (Relation).

ToDo: Transforms data flow. Application activation (available operations / state signatures / event flows). Browseable DOM / reified transforms (HATEOAS), reactive / event driven dialog / protocol: render applicable transforms.

Monad Transforms (hierarchically implemented):

Type Constructor.

Unit.

Bind.

Map / FlatMap.

CRUD (parametetized flatMap):

Entity::Append(Relation arg) : Function<Entity, Entity>;

Entity::Retrieve(Relation arg) : Function<Entity, Entity>;

Entity::Replace(Relation arg1, Relation arg2) : Function<Entity, Entity>;

Entity::Remove(Relation arg) : Function<Entity, Entity>;

Domain:

Relations / comparisons:

Invocation target: equals.

Argument: less than.

Distinct: greater than.

Entity::Assert(Entity | Relationship | Flow) : Function<Entity, Entity>;

Entity::Contains(Relation arg) : Function<Entity, Entity>;

Argument: contains.

Relation: statement / selector matching available arguments / possible operations (comparison results):

Entity::Activate(ctx : Relation) : Function[];

Entity::Available(Relation, Function) : Function;

Entity::Apply(Entity | Relationship | Flow) : Function<Entity, Entity>;

DOM Layers Monads:

Resource<OntResource[]>; Aligned Resources (metaclass, class, instance, role, occurrence);

Kind<Resource[]> : Aligned Types.

Statement<Kind[]> : Aligned Statements.

Relation<Statement[]> : Aligned Assertions.

Entity Monad. Aligned Resource (data):

Entity<Relation[]>;

Entity<Relation[]> Monad. Relation[]: selector, Relations which are instances of / wrapped by this Entity scope.

Example: Employment anEmployment Entity. anEmployment Relation Statements. Selector matches Employment Relations.

Entity Monad Transforms:

Apply Relation selector / Relation CRUD to Employment anEmployment.

Relationship Monad. Aligned schema:

Relationship<Entity[]>;

Relationship<Entity[]> Monad. Entity[]: selector, Entities which are instances of / wrapped by this Relationship scope.

Example: Employmentship aggregation / concept of Employment Entities. Selector matches Entities Relations.

Relationship Monad Transforms:

Apply Entity selector / CRUD to Employmentship Employments.

Flow Monad. Aligned behavior:

Flow<Relationship[]>;

Flow<Relationship[]> Monad. Relationship[]: selector, Relationships which are instances of / wrapped by this Flow scope.

Example: SalaryRaised Flow for Employmentsip Employments. Selector matches Relationship Entities Relations.

Flow Monad Transforms:

Apply Employmentship selector / CRUD to SalaryRaised Employments.

**Backend Architecture (To Do):**

XML: XSL, XPath, XLink, XPointer, XQuery, XForms: hypermedia addressing / state / flows encoding / Message endpoints protocol.

Models.

Messages.

Encoding.

Endpoints.

Protocol.

Domain Connectors.

APIs.

Models:

Dispatch to Model layers context resources streams. Message IO. Endpoint Message matches in Model context: activation (Dataflow).

Messages:

Meta Meta Model entities (Relation). Meta Model layers scoped context statements.

Encoding:

Model layers scoped context statements.

Endpoints:

Model layers context resources streams (pub / sub topics). Topics: OntResources(s). Signatures: Dataflow Message IO wiring.

Protocol:

Message Dialog: i.e.: XML encoded Context statements Message IO with Model layer scoped prompts, placeholders, wildcards, variables. Models. Messages. Streams.

Domain Connectors:

Tools. Service Resource: URL, streams (Messages I/O).

Tools:

NakedObjects / Apache Isis. Apache MetaModel. JBoss Teiid. JDBC. JCA. Apache Stanbol. Apache Clerezza. OData / OpenAPI. JSON-LD. Spring HATEOAS / HAL. Apache Any23. D2RQ. R2RML.

Message Matching:

FCA Augmented Models: Context objects / attributes: layer quad resources Role, Kind, Resource.

Resource Context Concept: Resource x Kind.

Resource Context Object: Role x Resource.

Model / Schema Matching:

FCA Resource Context Concept.

Data / Resource Matching:

FCA Resource Context Object.

Matching Productions:

Result Set (query / augmentation result).

Augmentation (aggregation of new statements, alignment of new knowledge, activation of transforms / flows: result set).

Relationship / Entity monads mappings results.

FCA Scaling: Role > Kind > Resource aggregation of matching objects / attributes.

FCA Augment Resources: Role, Occurrence, Object (Resource), Concept, Object, Kind. Grammars. Match schema, instances. Mapping transforms: match behaviors.

FCA Contexts from Sets aggregation:

Set Roles: Context, Subject, Predicate, Object.

Sets aggregation: Statement, Kind (SuperKind, Kind, Attribute, Value) Attributes: class / Values: metaclass, Resource (Meta Model Roles: Kind context, Resource SPO), Context (Relation). Reified Kinds.

Aggregation streams: Sets reactive events aggregation. Sets (ordered) description APIs.

Sets aggregation: FCA Contexts scaled objects / attributes from Sets aggregation. FCAAPI.

**Dataflow. Components:**

Uniform Resource API: Sets, FCA, DOM layers, Monads. Reactive message driven dataflow (topics / signatures).

Inputs. Connectors / Services (active Resource topics).

Sets aggregation.

FCA Scaling. FCA Contexts (layers / occurrences).

DOM Layers / OntResource hierarchy. Augmentation, alignment, activation, matching. FCA alignments (concepts).

(Sets aggregation populates DOM layers FCA augmented or Sets aggregation builds FCA contexts rendered into FCA augmented DOM layers).

Functors. Parse DOM: Instantiate Relationship / Entity Monads (selectors / contexts). Model services interactions renders functors possible transforms as browseable (HATEOAS reified) resources / contexts: reactive dialogs / prompts (HATEOAS / HAL protocols).

Model Services: Browse DOM layers. Monads parsed DOM interactions services (functor contexts) available as operations over rendered models (HATEOAS).

Interactions: Services. Browse DOM. Apply selectors / browse available transforms (Monads / HATEOAS). Monads applications render / update DOM / HATEOAS browsing response.

Outputs. Connectors / Services (active Resource topics). Feedback (Events Inputs).

**Deployment:**

Apache MetaModel. JBoss Teiid. Connectors (I/O). APIs: Model Services (reify data, schema, behavior alignment in Connectors data structures). OpenRefine Knowledge (data, schema, behavior) alignment extensions (Model Services APIs). Knowledge transactions (inferred "wizards") contextual wiki like augmentation: Apache Stanbol (guided assistance). Context Wiki: JCR XPath / Apache Clerezza.

**APIs:**

Contexts (DCI / HAL / HATEOAS):

Context Guided Data augmented (contextual hypermedia) Interactions. Wizards APIs. XForms: rendering (REST HATEOAS).

**Service Resources:**

Service Resources. ContextResource scoped prediction generalizations (encodings):

Meta Meta Model: (Relation, Statement : Entity, Kind : Relationship, Resource : Flow);

(PredictionService : Relation, Context: Entity<PredictionService>, Features : Relationship<Entity>, Output : Flow<Relationship>);

Naming Service (synsets, generalization/specialization term rels):

(NamingService, Context, TermRel, Term);

Registry Service (hierarchical key / value):

(RegistryService, Context, Key, Value);

Index Service (Apache Lucene, Vector Space Model Triple / Quad polygon encoding):

(IndexService, Context, Term, Result);

IOService (Connectors):

(IOServiceConnector : Relation, ContextResource : Entity, Attribute : Relationship, Value : Flow);

Relation: Aligned Context.

Entity: Aligned Resource.

Relationship: Aligned Schema.

Flow: Aligned Behavior (schema resource data flows).

ToDo: Resource selector API.

ToDo: Transforms data flow. Application activation (available operations / state signatures / events). Browse (HATEOAS), reactive / event driven protocol / dialog. Applicable transforms (example: IO Connector updates).

**Upper Ontology. Grammars. Primitives:**

Primitives:

Primitive Resources for primitive Kinds for primitive Statements for primitive Relations.

Primitive relations example: opposite, inverseOf, causeEffect, etc.

Grammars:

Resources productions for Kinds productions for Statements productions for Relations productions.

Upper Ontology: Primitives Grammar.

**Ontology Matching:**

Data, Schema, Behavior matching / alignment.

Data: keys / values.

Schema / Information: relation tuples rows.

Behavior / Knowledge: relation tuples rows data / information flows (dimensional).

FCA Augmented Resources.

See Protocol Addressing / Matching.

**Protocol. Message IO. Dataflow:**

DOM Hierarchy:

Resource<OntResource[]>;

Kind<Resource[]>;

Statement<Kind[]>;

Relation<Statement[]>;

Entity<Relation[]>;

Relationship<Entity[]>;

Flow<Relationship<Entity[]>;

Addressing / Matching:

Message Event dispatch. Message is an address plus browse state representation.

Matching. Distributed Addressing Encoding:

Encoding: Encode metaclass, class, instances, occurrences (contexts) in addresses.

Identifiers assigned according context objects and metadata "paths" following a pattern or "shape" in a way analogies can be inferred (Monad Zippers) of its (nested / linked ) metaclass, class, instances, occurrences "trees" (cons cells).

Matching: Same objects resolve to equivalent addresses in different models when addresses follows / match Zippers shape.

Populate ResolutionService Relations. Zippers paths / shapes specialized / generalized matches (concepts hierarchies). Others services metadata.

ResolutionService (Matching):

(ResolutionService : Relation, Resource : Entity, Model : Relationship, Resource : Flow);

Resulting Flow: next state (activation) over browse representation request.

Encoding / Parsing:

Representation: Message Event production / consumption.

Abstract Content Type: Render Model DOM Hierarchy.

Browse: request address content representation (extracted from current state) embedding current state representation as request context body. Model matches address and returns augmentation using request body as argument / context.