**Implementation:**

Definitions: Quads, contexts, Kinds, Grammar, etc.

Implementation: Runtime. Architecture. Components. Patterns. Models. Messages. Augmentation. Events dispatch. Message aggregation / instantiation / resolution / application. Backends. Services.

Resources inputs / outputs: Augmentation, Protocol, Browser. Message addressing / resolution / application.

Component: Models (data). Source: augmented input statements. APIs (Model).

Component: Messages (contexts). Source: augmented models templates (Grammar). APIs (Model).

Component: Transforms (interactions): Source: input statements case matching Message inputs. Returns / materialize results. APIs (Model).

Core Model API: Augmented (Aligned, Activated, Aggregated inputs matching model context messages) IO. Resource MetaGraph. Dimensional model. Grammar. Model repository. Backend. API.

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

Message: Context Model API. Input statements: Model Grammar. Augmented IO by interaction transforms of applied matching Message with model statements inputs. Context of core models instances. API.

Transform: Interaction Model API. Input statements: Transform request invocation specification. Functional application of Message(s) over Resource(s): Transform (streams). Augmented IO: Requested Transform which applied augments resulting responses (dialog arguments resolutions). Context of context model instances. Reactive / streams API.

Message Transform (interaction result): matches request context specification built upon Resources / Messages (TransformBuilder). Resolve state / dialog session graph. Returns observable stream. Dataflow (chaining). Operations (over streams).

Transform request invocation specifications: means to interact with underlying contexts models (CRUD, domains behavior). Transforms result from applicating Message(s) over Resource(s). Sending a Message Resource to a given interaction context initiates a “dialog” in which to “populate” target Resource(s) and Resource arguments. Each dialog “step” renders resources / layers streams of requested arguments (server “queries” clients) or resources / layers streams of response augmented Resource(s).

Message kinds IO (Augmented async request / response):

Data Message: Assertion (attributes / links).

Schema Message: Types (kind / class).

Behavior Message: Interactions (flows / behaviors).

Specification resolves to query / create / update / delete according interaction contexts.

Interaction Model: Context of Messages model for a given interactions session / dialog state. Message invocation requests: Statement(s) building Resource invocation graph with layers matching Message patterns. Layers graph invocation patterns matching from higher to lower layers resources fulfilling higher layers templates. Variables, wildcards, placeholders.

Dialog arguments resolutions example: higher layer Resource / Message request / invocation instantiates in Interaction Transform context corresponding lower layer graph statements to be “populated” to fulfill request. Message IO of “forms” (Messages) inter-peers (originating peer acting as “server”) for inititial requested peer to “ask” for form elements to be populated (interaction context “dialogs”). Resolution may propagate to other peers (content aware addressing dataflow routes dispatch: P2P resources address encodings, matching forms models requests). Nested interactions.

For input Resource(s) (Model reactive / async IO APIs):

. Create / retrieve Model

. Create / retrieve Context Message(s)

. Create / retrieve Message(s) Interactions

. Bind Interaction Message Resource(s)

. Perform Message transform. Materialize results. Message application rules: upper / domain ontology selectors (closest matching role in hierarchies), context alignments.

Services: distributed addressing / resolution, reactive distributed event bus: streams / contracts, index, naming, registry.

Discovery: All model kinds are browseable / discoverable.

///

**Purpose driven hypermedia activation:**

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

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

Browser referring context (Work, Peter, Employee).

Models ‘plug’ into Runtime augmenting its capabilities via standard extension APIs (added features / knowledge).

Models ‘modules’: parsing modules declarative descriptions. Augment, link instance data.

Upper aligned ontology plugins / blueprints.

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 and schema.

///

**Resource Activation Protocol**

Annotate, link, browse resources instances, classes, metaclasses, occurrencesin roles in contexts. Services / clients: endpoints: Virtualization (wrapper protocols).

Semantically annotated content types: image/png;face, text/xml;faceImgCoords. RDF schemas describing content, attributes, links in context / target roles.

///

**Messages Metamodel (Context Model):**

Message types (Augmentation: onto / domains):

Attribute / Link (data):

. Alignment: Augment / infer Attribute / Link.

Class / ID (schema):

. Activation: Augment / infer Kind, Class.

Role / Context (behavior):

. Aggregation: Augment / infer Role / Context.

Runtime / Resources / Messages: Core (upper / onto) Resources, Messages, Transforms. Reified entities (CSPO, Kind, SubjectKind, etc.). Match cases in messages.

Core (upper / onto) Messages: Getters, setters, nav, etc.

Domain Messages: raiseSal: setSal(sal \* increment); promotion: setPosition.

Event sourcing / tracking: married -> marriage occurred.

Resource.flatMap(messageInst::apply) : Resource.

Dataflow: Messages hierarchy. Aggregate contexts from coarse to fine grained transforms (raiseSal -> setAttr).

data <-> schema <-> behavior.

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

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

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

Activation Message: Statement -> Kind, Class.

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

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

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

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

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

Processor which acts upon Resource events. Materialize results.

Specify declaratively augmentations by means of messages.

Upper onto / domain aggregated messages.

Event bus: P2P deployment.

Messages: Monadic applicables over Resource (flatMap).

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

///

**Addressing: URIs, DIDs URLs. Adddress, content type, representation (URI APIs). Browse / CRUD (DAV).**

Resource<T : URI> monadic hierarchy. Basic hypermedia browse / CRUD (HTTP verbs) bound Message functors compatible for all Resources (REST).

Resource.flatMap(Message::apply) : Observable<Resource> (stream). Composable functions.

Basic Message application (Context Mapping): shift right mapped applied statement resources. Mapped resource context> instance of mesage reified resource context.

Transforms (Message templates):

Resource: Statement

Message: SubjectKind

Transform: Resource

Resource: Statement

Message: ClassLayer

Transform: Statement (class)

Resource: Employee

Message: Position

Transform: Manager

///

**Resource API**

Resource<T : URI> monad. Message functors. Transform reactive extensions.

Transform : Observer / Observable of Resource<T : URI>. Stream. Built upon Resources / Messages (TransformBuilder). Identity and other core transforms (core messages). Stream. flatMap(Message::apply) : Transform<Resource<R : URI>>.

API: Class for layer for model.

API: Class for layer (DOM).

API: Parameterized Resource: layer classes determined by URIs hierarchy, i.e.: Resource<Entity>, Entity : URI.

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.

///

**Quads reference model. Kinds. Grammar.**

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

Aggregation.

Kinds.

Grammar.

Formalization: Functional / Object API. Reference / Data model. Sets, categories, models.

Subjects: attributes / values, contexts / roles.

(Context, Occurrence, Attribute, Value);

(Context, Sign, Concept, Object);

instance, occurrence, class, metaclass.

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.

///

**Metamodels: Graphs** (Models)

Composed of quads semantically aggregated into layers.

Core features provides:

Alignment

Activation

Aggregation

Message / Transform driven specification of Alignment, Activation, Aggregation.

Grammars.

Upper / Dimensional ontology.

Inter models alignments

Services (Endpoint URIs: Resource facades).

///

**Encodings**

aX\*4 + bY\*3 + cZ\*2 = dW

d, a, b, c: classes (CSPO)

WXYZ: instances (CSPO)

powers: CSPO role

terms: CSPO resources

Z(obj) is Y(pred) for X(subj) in W(ctx)

instance, class, metaclass, ocurrence terms.

primitives, variables, placeholders.

resolution (Discovery, DIDs).

Templates (grammar)

Subjects: attr / val, ctx / role

Behavior: order / compare.

Proof of work

MetaGraph model: map URIs -> IDs

Satisfy dW. Sync resolution (recurse terms contexts)

FCA. Resource attributes. Tensor, adjacency matrix, tree.

///

**Reactive Dataflow. Resource / Message / Transform. Behavior in graphs.**

Message flow (event loop) in / out: Alignment (data) <-> Activation (schema) <-> Aggregation (behavior)

Encode behavior in statements / graph:

Comparisons, order. Sort. Order (kinds hierarchy?)

Pattern matching, iteration, jumps. Discovery: routes / signatures, next event in bus / graph.

Context Model Message: Resource Specification (Grammar Template). Messages Model: context model instance from input model grammar. Transform: context model instance from Messages.

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

///

**DCI Metamodel (Base Model)**

(Entity, Statement, Attribute, Value);

(Kind, Entity, Statement, Attribute);

(Class, Kind, Entity, Statement);

(Flow, Class, Kind, Entity);

(Behavior, Flow, Class, Kind);

///

**Dimensional Metamodel**

(Value, Distance, Previous, Next);

(Measure, Value, Distance, Previous);

(Unit, Measure, Value, Distance);

(Dimension, Unit, Measure, Value);

(Concept, Dimension, Unit, Measure);

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

Assert knowledge: 1h -> 60min, lun-mar-mie-jue-vie, 1mt -> 100cm.

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

Sort: cause / effect, temporal, etc. Messages align, functional map, fold, etc. Primitives.

Encode layered statements ordering. Complement / suplement concepts definitions.

Events metamodel (TBD):

(Object, State, Axis, Type)

(State, Axis, Type, Event)

(Axis, Type, Event, Event)

(Type, Event, Event, Event)

(Event, Event, Event, Event)

Model MetaGraph (TBD):

(ResourceClass, ResourceID, Statement, Kind);

(StatementClass, StatementID, ResourceID, Kind);

(KindClass, Kind, StatementID, Kind);

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

Events declarative definition. State change of value in axis in measure of context.

Messaging metamodel:

(Message, Resource, LHS, RHS);

(Interaction, Message, Resource, LHS);

(Role, Interaction, Message, Resource);

(Context, Role, Interaction, Message);

(Dataflow, Context, Role, Interaction);

Domains metamodel (TBD?)

///

**Event sourcing (“offline” sync). API**

Graph linking / alignment / sinchronization by entailments from event sourcing over inferred state.

DOM / OGM APIs (JAF).

I/O Implementation, Deployment.

Model, URI, Resource, Statement, Kind hierarchies. Models architecture (URI class per layer).

DIDs / P2P / Rx Implementations.

Model API. ModelManager.

Event loop. IO.

Protocol: Input statements for querying augmented knowledge. 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.