**Distributed Consistency for Semantic Integration of Applications Knowledge**

**0: Outline**

**0.1: Mision**

**0.2: Vision**

**1.: Use Cases**

**1.1: Problem**

**1.2.: Solution**

**2.: Approach**

**2.1.: Augmentation Dataflow KB**

**2.2.: Ontology Matching**

**2.3.: Reactive Event Driven Architecture**

**3.: RDF: Introduction**

**4.: RDF Quads / Object Mapping (OGM)**

**5.: Models**

**5.1.: Contexts Quads Layers**

Kinds: example aggregate CSPO Kinds from an Statement. Kind / Class hierarchies. Order (dataflow / dimension / grammar / facets).

Class intension / extension (Context / Value).

**5.2.: Meta Resources**

Meta Model Context hierarchy.

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

ID<ID> : Reified matching URIs.

Transform<ID> : Range

Mapping<Transform>

Template<Mapping> : Domain

Augmentation<Template>

Resource<Augmentation>

Role<Resource> : CSPO Role

Statement<Role> : CSPO Quad

Model<Statement> : Set of Statements

**5.3.: Meta Model**

Messages: Augmentation (performed transform), Template, Mapping (possible transform), Transform.

Reactive Uniform Component API: Context reactive instances: network addressable / operable (pub / sub streams). Wrapper API.

Addressing: Encoding. Network URLs, Semantic URNs. Naming, Index, Registry operations.

Dataflow Routes: Context Signatures. Forms. Bus. Addressing dispatch resolution.

Message: IDs Statement. Specifications (CRUD).

Template Matching: Alignment / Encoding. Populate Template with Message. Map.

Augmentation Mappings: Flows (Wrapper API). Exchange.

Materialize Specification: Transform (Wrapper API). Reduce.

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

ID<ID> : Reified matching URIs. Message (encoded reified verb?)

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 Role Resource Occurrences

Messages: Augmentation (performed transform, Flow), Template, Mapping (possible transform, Form), Transform. Dialog.

Model Reactive I/O:

Model forward (map inputs): aggregate inputs into reified layers contexts instances (Model Meta Resources reification).

Augmentation: populate / perform Flows. Aggregate, Align, Activate (over mapped inputs). Mapping Template Transform algorithms / services encoding in Statement plus Meta Resources.

Model backwards (reduce outputs): collect occurrences graph (matching signatures contexts from Model layer to IDs).

ContextClass<OccurrenceClass> : Layer (IDs). Attributes, Values.

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

Layers:

Resource?

(ID, ID, ID, ID); Message (encode reified verb?)

(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); Resource Occurrence in Model (Kind? Kind Role type, Kind hierarchies.)

Functional layers?

Resources (reactive entity: quad / ID):

Layers Context types: reified Resource quads instances (Meta Model Kinds). Resource quad wrapping: signatures bindings.

Context layers instances: Meta Model Kinds hierarchy. Resource quad wrapping: signatures bindings.

Meta Model Kinds hierarchy (super / sub Resource class / kinds rels: super: ctx kind, sub: obj kind). Reified model entities: layers super types,

Uniform Reactive Resource Quads Wrapper API (Resource / IDs / Message / Layers): Metaclass (P) / Class (C) / Instance (O) / Occurrence (S) CSPO Resource roles / rels members, monads / transforms. Events domain / range.

Reactive component: Resource, pub / sub (endpoints APIs) for wrapping signature bindings (layers). Events domain / range.

Aggregations: Subject / Attributes, Attributes / Object, Subject / Object (Kinds). API for Functional layers interaction / composition.

Enable uniform treatment of Resources for layers aggregation / augmentation, etc.

Message / Specifications (Mappings Forms / Flows). Encoding (Specification, Form, Flow) of Mapping Transforms.

Transform. Compare: common upper types.

Encoding: nested shapes of recursive cuads (till primitives). Patterns / expressions: wildcards, variables, placeholders:

[[123, 456, \_b, $a][\_b][\*][$a]]

Resource Component:

reactive resolution / instantiation events matching conditions.

Resource class / component kind:

Members. Relations: Graph quad layers bindings (DOM). Previous, next, parent, child (order: class hier relations), Resource (instance), Role (metaclass), Statement (occurrence), Kind (class). Resource Monads. Eval rels axis: instance. Functors: ID Monads rels traversal.

(Context, Occurrence, Attribute, Value);

Events API. DOM. Monads. Functors (domain / range). Relations / traversal. Events. Encoded in Meta Model (Message Monad).

Encoding: signatures / bindings. Representation query / traversal / transform. Class / instance Functor / Monads relations Dataflow.

Resolution / instantiation: Resource & Resource members / graph quad layers bindings.

Statements: Backend. Encoding. Addressing. IDs. Sync Functional Object Model. Services. Reified Object Model. Onto aligns: upper / dimensional. Messages I/O: IDs matching / alignments.

Functional Object Model:

Events (Message I/O) conditions matching (resolution / instantiation / bindings).

Objects: componed of aggregation of monad resources of a reified ID.

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

Forms: (Template (Mapping (Transform (Augmentation))));

Object: (Value (Attribute (Occurrence (Context))));

Members, relations, endpoints (API).

Message I/O: Dispatch according signatures bindings. Augmentation events (Functors) Kind streams.

Message<Monad<ID>>: Objects matching Message encoded structure (value, attribute, occurrence, context, class / instance, rels). Resolve, instantiate: perform Message logic (encoded in Model monads / Meta Resources). Example: Map Reduce. Return Dialog Graph Message.

ID

Message<Monad<ID>>

4 Transform: Range Flow

3 Mapping: Bind Specification

2 Template: Domain Form

1 Augmentation: Verb / Event. Browse rels

4 Resource

3 Role

2 Statement

1 Kind: Event streams

(...Model, Functional? Class, Entity, Flow, Behavior)

OGM: Object Graph Mapper. REST Facade. Domains activation.

**5.3.1.: Facets**

**5.3.1.1.: Functional Facet**

(Statement, Role, Resource, Augmentation);

(Entity, Statement, Role, Resource);

(Kind, Entity, Statement, Role);

(Class, Kind, Entity, Statement);

(Flow, Class, Kind, Entity);

(Behavior, Flow, Class, Kind);

**5.3.1.2.: Semiotic Facet**

(Context, Sign, Concept, Object);

Object as Sign: Concepts represents attributes (DOM / OGM). Ontology Matching (shapes).

**5.3.1.3.: Dimensional Facet**

(Value, Previous, Distance, Next);

(Measure, Value, Previous, Distance);

(Unit, Measure, Value, Previous);

(Dimension, Unit, Measure, Value);

(Concept, Dimension, Unit, Measure);

**5.3.2.: Layer Levels**

Levels: reify Meta Model Contexts hierarchies into IDs Layer.

Context<ID>; Signature route. Reactive producer / observer.

Mapping<Context<ID>, Context<ID>> : Context<ID>;

Mapping<Template : Person (S), Transform : Address (O)> (P) : Context<ID> (C) : Augmentation;

Mappings Encoding: parse Template in context of Transform. Augmentation: materialized result Resource (query / assertion).

Meta Model: IDs to Context hierarchy Mappings.

Facets: Context to Model Mappings. Data / schema / behavior class / instances views (aggregation) APIs.

Meta Model levels reification populates / resolves Mappings.

Queries / Assertions: Domain driven and Core Augmentation Messages: Model Message layout (Flows browseable API) defined in models levels reifications (grammars / layers / facets / levels).

Model Message layout Dataflow: Mapping routes, Templates, Transforms signatures matching (bindings).

Model Message layout Augmentation: Message input transform / alignment (raw quads: ontology matching / match Model patterns). Message<Context> : apply Dataflow transforms. Emit resulting Message (dialog / feedback).

**6.: ID: Context Reactive Abstraction**

Model IDs. Service (Connector / Client) IDs. Addressing reactive abstraction.

**7.: Encoding**

Model Encoding: Property graph. Properties (prefix codes, key / value, reification). Sets, groups, categories. Functors applications: Transforms as graph navigation / browse. Template Message parsing (grammar, verbs, state flow). Contextual Quad Context ID: ID according occurrence in Statement context (normalized forms). Occurrence Context IDs indices / mappings.

**8.: Signatures**

CSPO Context Kind (Statement Subject Kind + Object Kind). Context Dataflow domain / range (Context as reactive streams producer / consumer).

**9.: Routes / Dataflow**

Routes: Dataflow pub / sub bindings between matching signatures.

Core Model and Domain driven Message flow layout (Mappings).

**10.: Event Bus**

Dispatch Event into Dataflow Route.

**11.: Model I/O**

**11.1.: Mappings: Events Transforms Declarations**

**11.2.: Layers / Facets Transforms**

**11.2.1.: Aggregation**

**11.2.2.: Alignment**

**11.2.3.: Activation**

**11.3.: Augmentation: Events Mappings Realizations**

**12.: Backend**

**12.1.: Model Containers**

**12.1.1.: Services / Protocols**

Layers (session, dialog, etc.). Node, Peer, Client, Connector, etc. Reactive / Event Driven. REST HATEOAS.

**12.2.1.: Models Declarative Encoding**

**12.2.2.: Functional APIs**

**12.3.: Persistence**