Application:

Workflows (Domain Goals) general purpose ontology matching integration framework.

Component Model:

Reference Model Message Encoding (Component Node types Message Adapters). Component Monads of Component Node types: Functional reactive stream / events pipelines Bus dataflow (selector signatures).

Bus:

Reference Model encoded Messages. Component Monad of Node types instances pipelines: Node type handler of Message I/O translation / consumption / production). Messaging backend.

Component.of(node);

bus.connect(component);

(for bus.components)

Component next = component.flatMap(NodeType::consumeCurrentMessage);

(next until end of depth / list of nodes)

Bus next: enqueue produced messages. Reactive streams. Pipelines.

Reference Model:

Encodings.

Component Nodes:

I/O:

I/O. Persistence. Events (event sourcing). DIDs Components Nodes.

Sets:

Component Node.

Triple Store:

Layers Quads Component Node.

Meta Model (Quad Layers / DOM):

CellValue

ColumnField

ID : occurrence (PK)

Context : instance (table)

Role : metaclass (CSPO)

Resource : class. Monad Value (instance)

Kind : selector / transform (Functor mapping). Monad Value Type (metaclass / role)

Statement (context)

Relation : Kind Grammar (Productions). Monad Instance (occurrence)

Entity : Kind Grammar (Rules). Monad Type (class)

Relationship

Flow

Domain

FCA:

Component Node.

FCA Ontology Matching: Upper ontology / primitives. Reference Model objects / attributes encoding. Encoding (scaling): lattice concepts relations / transforms traversal.

Semiotic mappings population. Augmentations: Aggregation (layers), Alignment (ontology), Activation (layers dataflow transforms: context products).

Functional APIs:

Component Node.

Monads AST / Parser Builder.

Layers Monads / Parser Monads (Messages : Rules / Productions). Functional events dataflow (selector signatures : Layer instance Activation) Component Node.

DOM / AST Hierarchy:

Resource<OntResource[]>;

Kind<Resource[]>;

Statement<Kind[]>;

Relation<Statement[]>;

Entity<Relation[]>;

Relationship<Entity[]>;

Flow<Relationship<Entity[]>;

Domain<Flow[]>;

Layer production = Layer.of(resource);

Layer rule = production.flatMap(ResourceType::matchRule);

Layer.of: Return matching hierarchy context type.

Resource::match: Production Rule.

Resource::matches: Rule Productions.

Layers: Productions in one layer are Rules of next layer (context, metaclass, instance, etc.).

Message as Resource (prompts).

Endpoints:

Forms / Flows (Grammar / Protocol Builder. Prompts) Component Node

Augmented Resources Contexts / Interactions Services Component Node

OGM / Client Drivers Services Component Node.

Ontology Matching:

Services / Mappings:

Upper Ontology. Grammars. Primitives.

FCA Ontology Matching.

Matching: Resource occurs as context / occurrence / atribute / value or class / occurrence / context / metaclass / instance in equivalent occurrence contexts (kinds / order / shapes / type hierarchies).

Meta Model encodes mappings for equivalence / relations hierarchies for entities instance occurrences in roles in contexts for concepts recursively till upper onto / primitive terms / relations.