Semantic Reactive Microservices

Encodings

The whole idea revolves encoding 'roles' (Resource types) in an aggregated dimensional layered (data, schema, behavior) scheme via the use of 'kinds' abstraction.

Kinds abstraction:
sets / type inference. Given an statement SPO, for example the Subject, its 'subject kind' is the aggregation of all Predicate / Value pairs occurring in statements having that Subject. Subjects having a common set of this pairs are regarded as having the same 'kind'. Subset / superset relations between those pairs determine super / sub 'kind' relations.

If we only regard of Predicate occurrences for a common Subject we can have a very basic class / type inference also with a predicates subset / superset (class / superclass) relation. The same holds for 'predicate kinds' and 'object kinds'. A class could be, for example, Person and kinds (metaclass) of occurrences of this class could be: Employee, Father, etc.

Resources hierarchy (static CSPO roles):
(Statement : Entity) : Resource;
(Kind : Class) : Resource
(Flow : Behavior) : Resource

Resource hierarchy (CSPO instances):
Resolve by kinds type inference. Example: Kind: super / sub Kind, Class: super / sub class, etc.

Resource API:
Dataflow (Resource I/O, activation: observer / observable). Role type hierarchy and Resource type hierarchy.

Abstract Layers:
Dimensional ontology aggregated CSPO roles / statements. Resource layers maps / corresponds into this statements schema / forms for diverse augmentation (aggregation, alignment, activation) use cases.

(Dimension, Unit, Measure, Value);
(Context, Concept, Sign, Object);

(Context, Occurrence, Attribute, Value);

Resource Layers:
Hierarchically aggregated statement layers.

(Entity, Statement, Attribute, Value);
(Kind, Entity, Statement, Attribute);
(Class, Kind, Entity, Statement);
(Flow, Class, Kind, Entity);
(Behavior, Flow, Class, Kind);

Upper ontology layer:
Statements aggregated from data, schema, behavior layers statements (used for Contexts I/O).

(Behavior, Class, Entity, Resource);

Resource encoding: (C, S, P, O) : Resource. Resource IDs.

Resource representation:
Render Resource in CSPO role in statement occurrence. C(S, P) : O.

Functional Forms:
Resource encoded code and data. Parse representations, apply transforms. Syntax (grammars, wrappers, monadic parsing).

Metaclass (behavior, role) / Class (schema) / Instance (data).

Event (definition): measure value change in dimension for unit.

Activation:
Resource I/O events stream (quads). Materialize new / inferred knowledge, emit known facts regarding event.

Aggregations:

Object occurrences aggregates into Predicate occurrences which aggregate into Subject occurrences and then in Context occurrences.

Entity: Plain RDF URLs input.
Subject example: (S, SPO, P, O);

Kind: Entity occurrences aggregation.
Subject Kind Example: (Kind, Entity, P, O);

Class:
(Class, Kind, SPO, Entity);

Flow:
(Flow, Class, Kind, Entity);

Behavior:
(Behavior, Flow, Class, Kind);

Upper:
(Behavior, Flow, Kind, Class);

Contexts

Resource wrapper:
Aggregated internal layered (data, schema, behavior) quad statement sets. Behaves as a data, schema or behavior layer via activation composition (Resource I/O).

Interacts (I/O activation) via upper ontology layer aggregated form facade statements:
(Behavior, Class, Entity, Resource).

Resource IDs / namespace handler.

Index, Naming, Registry facades.

Addressing / annotation (augmentation) resolution of external resources. JAF.

Augmentation: Aggregation, Alignments, Activation.

Reactive streams: events, locators, filters, transforms, queries, aggregation, getters / setters, joins, etc.

Functional Forms: code / data stated as resources. System resources. Bound functions / transforms.

Layered aggregated contexts stack:
(Context (Application (Domain (Data))))

Functional Forms: Browse layered aggregated context (render services / applications).

Layers integration: dimensional resources 'overlay' interacting via activation of upper ontology layer aggregated form statements.

Data contexts: data, schema, behavior aggregated I/O from plain RDFized (Resource IDs) inputs.

Domain contexts: data, schema, behavior aggregated I/O from Data contexts aggregated upper ontology.

Application contexts: data, schema, behavior aggregated I/O from Domain contexts aggregated upper ontology.

Resource IDs

The idea is to achieve a (numbering) identification scheme which allows to encode and identify RDF statements CSPOs URLs (and the URLs referring to the statements itselves) in a manner which:

a) Allows to ‘embedd’ meaning in an algorithmically 'operable’ way.
b) Enforces preservation of 'validity’ between identifiers (no non-valid identifiers could be forged)

In base to the abstract layer (semiotic) statement form:
(Context, Concept, Sign, Object);

The idea is that (in theory) using a positional ternary numbering system with a cyclic order relation (a > b > c < a) CSPO IDs could be validated against:

C > S < P < O

For any given statement IDs arrangement.

Primitives:
self > this > that < the

Alignments:
X is Y of Z in W;
C(S, P) : O;

ML Embeddings.

ID Assignment

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

S : Concept, P : Sign, O : Object.

(the, self, this, that);

Context::nextID(URL / ID, leftLastID, currPosLastID, rightLastID);

C : nextID < S : nextID > P : nextID > O : nextID;

Functional Forms

Functional code / data serialization format / language expressed in terms of Resource statements.

TID: Statement Context ID.
VID: Statement Subject (Occurrence) ID.
TID:VID: Context / Occurrence attribute / value sets (recursive forms).

Form:
(TID:VID (TID:VID (TID:VID (TID:VID)))) : TID:VID;

(Behavior (Flow (Class (Kind)))) : Dimensional abstract Resource (Entity);

Assertion / query language. Activation.
Algorithm resolves over Behavior, Schema, Data attributes / values.
Dataflow activations: candidates for resolution (signatures / injection).
Specific system forms (augmentation bound functions).
DOM / LINQ like APIs.
ML Embeddings.

Functional Activation

(Context, Occurrence, Attribute, Value);

(Entity, Statement, Attribute, Value);
(Kind, Entity, Statement, Attribute);
(Class, Kind, Entity, Statement);
(Flow, Class, Kind, Entity);
(Behavior, Flow, Class, Kind);

Form:
(TID:VID (TID:VID (TID:VID (TID:VID)))) : TID:VID;

(Behavior (Flow (Class (Kind)))) : Dimensional abstract Resource (Entity);

Mappings:
Resolve Flow from Behavior, Class from Flow, Kind from Class, Entity from Kind via mapping matching:

(O -> P); (P -> S)

From lower layers to upper layers.

Features

DCI: Data, Context, Interaction.

(Context, Interaction, Data:role, Data:state);

Data (event), Information (flow), Knowledge (rule: context, role, state flow).

Type inference: Entity, Statement, Kind, Class, Flow, Behavior. Functional Form syntax, upper ontology.

Dataflow graphs: dynamic 'routes' (Resource stream observer / observable) 'signatures' (activation matching: resolution / injection). Aggregation (layers), composition (contexts), discovery.

Augmentation (ML / Type inf. / Dataflow):
Aggregation
Alignment
Activation

Alignments:
Class / ID
Attributes / Links
Contexts / Roles

RDF / OWL Backend.