This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

<http://creativecommons.org/licenses/by-sa/4.0/>

2018 Sebastián Samaruga

[ssamarug@gmail.com](mailto:ssamarug@gmail.com)

# 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:

1. Allows to ‘embedd’ meaning in an algorithmically 'operable’ way.
2. 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.

## Implementation (notes)

**Goals:**

Integration:

Integration of federated sources of knowledge and invokable behaviors (applications, services, backends, etc.) in a transparent and uniform manner enabling ease of distribution and composition into new aggregated assets.

Data Consistency in Microservices Architecture:

“In microservices, one logically atomic operation can frequently span multiple microservices. Even a monolithic system might use multiple databases or messaging solutions. With several independent data storage solutions, we risk inconsistent data if one of the distributed process participants fails — such as charging a customer without placing the order or not notifying the customer that the order succeeded.”

“Why is it so challenging to achieve this? As long as we have multiple places where the data is stored (which are not in a single database), consistency is not solved automatically and engineers need to take care of consistency while designing the system. For now, in my opinion, the industry doesn’t yet have a widely known solution for updating data atomically in multiple different data sources — and we probably shouldn’t wait for one to be available soon.”

(<https://dzone.com/articles/data-consistency-in-microservices-architecture>)

Integrity Validation

Encoding of data supporting “validity / consistency” checks enforcement of messages avoiding the possibility of “forged” or fake data without third parties authorities or ledgers.

**Tools:**

Microservices: domains common classes / interfaces (behaviors) : DOM / OGM Layer. Declarative description of original business (legacy addressing / activation).

API Gateway. Composite service (aggregator). Database (Context) per service.

Command Query Responsibility Segregation (CQRS). Events sourcing / views. Saga: compensating requests. Logs: aggregate standarized log format for each service instance.

Distributed tracing: external request unique ID. Sub requests contextual 'external' IDs (graph). Record logs.

Externalize configuration. Command pattern.

Service discovery: registry. Locate by interface / schema signatures (capabilities).

Java. Functional Resource API. DOM / OGM.

RDF / OWL Backend.

Declarative Resource REST API Gateways (I/O: aggregation, forms activation, etc.).

Reactive Framework. Container. Addresses (Resource IDs mappings).

Messaging. Dataflow. Routing (Resource IDs / Address mapping). Layers. Injection (Forms activation).

Augmentation: Aggregation, Alignment, Activation. (ID / Class, Attributes, Roles Alignments).

ML Declarative (functional) services API (general shapes / models). Naming, Registry, Index. Serving (activation shapes).

ML Context Index Service: dot notation graph encoding. Templates, grammars, activation. Facets. Class / ID Resolution.

ML Context Registry Service: attributes / links in contexts resolution.

ML Context Naming Service: dictionary: key / value (recursive) store. Synsets. Senses (contexts / from SPOs occurrences). Embeddings. Role in context resolution.

Domains (Upper ontologies) DOM, OGM Context metadata for: CMS, B2C, B2B, ERP, CRM, SCM, ESB, EAI, BI (business domains).

Endpoints Rendering: Declarative resource described / driven APIs. Microservices.

Clients Rendering: Declarative activation DCI / MVC contexts / flows. Portlets.

Adapters: JBoss Teiid. Apache Metamodel. OData. SoLiD. HAL / HATEOAS. GraphQL.

XML / XSL / XLink / XPath / XPointer.

JavaScript.

## To Do:

**Document:**

Schema: docs application model. Resources dataflow nodes graph. Activation.

Schema: docs data model. Resources encoding. Aggregation.

Schema: docs reference model. Resources arrangements. Alignments.

**Applications:**

Domains upper ontologies: DOM / OGM.

**Resources**

Context<? extends Resource> : Resource;

Monad of Resource hierarchy. Match / apply Templates / Transforms (bound functions). Streams (observable).

Resource: CSPO map: (this ( occurrences( attributes( values)))).

APIs:

Factory methods. Builders. Visitors. Enum for case classes. Functional architecture. Patterns.

Node: Connectors, I/O, services. Address resolution (activation events / messages) of Node domain Contexts (Resources). Backends.

(Context, Occurrence, Attribute, Value);

(Resource, Resource, Resource, Resource);

Application, data, reference models. Graphs. Sets, Groups, Categories models.

**Resources hierarchy (layers)**

Occurrence, Attribute, Value : Resource.

(Resource, Occurrence, Attribute, Value);

(Statement, Resource, Occurrence, Attribute);

(Entity, Statement, Resource, Occurrence);

(Kind, Entity, Statement, Resource);

(Class, Kind, Entity, Statement);

(Flow, Class, Kind, Entity);

(Behavior, Flow, Class, Kind);

**Role Kinds abstraction**

**Basic type inference**

**Instances hierarchy (Role Kind)**

**Specifications**

* Specification
* CSPO: Specification
* Specification::eval(ctx : Specification) : Specification
* Declaratively built: specify CSPO slots (specializations types and attributes schema) and 'eval' I/O behavior.
* Underlying RDF data model.
* Specializations (hierarchy, up until application description / rendering components):
* Message
* Node
* Resource
* Template
* Form
* Activation
* Aggregation
* Alignment
* IDs Graph Resources
* Address
* Connector
* Functional mapping (Activation Template query)
* Computations (ordered graphs)
* Ontology matching (Specification)
* Upper ontology
* Use Case layers
* Implementation: Java, XSL (runtime), JavaScript, XML (components). Functional DOM like API for resources / specifications setup / stacks "standard" events / callbacks driven integration.
* java.util.Stream (Resource). Reactive Streams (Observable, Publisher, Subscriber, Processor, Subscription).

**Templates: Resource specifications**

Template: map. Role Kinds match (types). Activation / aggregation. Node Resources traversal. Factory / Builder.

**Forms: Transform specifications**

Form: reduce. Role Kinds apply (flatMap). Activation / aggregation. Node resources materialization. Factory / Builder.

**Ontology Matching**

Specification : Resource (Context);

Node: Connectors. Resources feed. Custom URLs for different services / backends I/O.

Resource layer: Resource instance for each one CSPO roles in quad statement input, Specification for the rest of CSPOs in Resource layer statements. Example for Subject: (Specification, Subject, Specification, Specification).

Aggregation into Statement layer by Resource and roles Specification matching. Resource are instances from resource layer (with Specification in roles). SPOs in Statement occurrence role w./ attrs & values.

Refine Specification "learning" until matching Context Subjects, Predicates and Objects unambiguously from inputs.

**Functional mapping of role kinds to resources**

**Aggregation: Via Template / Form mapping**

Lower layers Resource hierarchy aggregates into higher layers by resource occurrences aggregation.

Rules (Template, Form) :

Entity: (S, SPO, P, O);

TBD.

Reactive event driven dataflow: matching Aggregation Template Form bindings define Reactive Streams “subscriptions” between involved Resource(s).

**Alignments: Services**

Services models:

Context / Role Alignment.

Attribute / Link Alignment.

Class / Instance ID Alignment.

Registry (clustering), Naming (classification), Index (regression) services models.

Embeddings, autoencoders, classifiers of contexts and inputs.

**Activation: Resource transaction**

Message Event broadcast driven transform by applying received Form over matching received Template. Transform result (new knowledge) broadcasted to the network.

Nodes post / react with these messages when a new entity is found and to query other peers about generalized / specialized knowledge regarding such entities.

Reactive event driven dataflow: matching Activation Template Form transforms define Reactive Streams “subscriptions” between involved Resource(s).

Smart contracts.

**Resource IDs: IDs Graph. Addresses**

Resources DID (W3C Distributed Identifiers) implementation. DID Documents: API contracts / HAL.

Unique Resource ID: URL.

Unique (distributed) sequence ID.

Forge Resource hierarchies URLs.

Forge Class / Kind URLs.

Layers Aggregation rules (from Behavior to Resource):

(C: prevOcc, S: nextCtx / prevPred, P: nextSubj / prevObj, O: nextPred);

Implement Specification (Template / Form) for previous rules.

IDs Graph (quads):

ResourceID: (ResourceClsID, ResourceInstID, StatementID, RoleKindID);

StatementID: (StatemenClsID, StatementInstID, ResourceID, RoleKindID);

RoleKindID: (RoleKindClsID, RoleKindInstID, StatementID, ResourceID);

Address:

did:ont:clsID:instID/clsID:instID/clsID:instID

Signatures (Specification): TBD.

Discovery: TBD.

Composite query expressions building Resource IDs Graph statements.

Aggregation (“counting”) of sequence IDs. Tree. Primitives (the, self, this, that). Next, prev, curr ID functions.

Encode orderings in class relationships. (via role kind).

Encode orderings in instances relationships (via role kind).

Encode orderings in class / instance relationships (via role kind).

**Messages: Activation documents (assertions)**

Event driven Resource Node(s). Blockchain transactions for each assertions set (Activation).

Addressing / annotations of resources (XSL, XLink, XPath, XPointer). Activation of content types.

**Upper ontology: Initial Resource Activation layer**

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

Root (IDs Graph hierarchies) Resources. Initial activations upon upper ontology.

APIs: Activation transforms / functional bindings. Use dimensional, grammar, syntax representations.

**Domain / Upper ontologies**

ERP Ontology, ISO, TMDM, TMRM, SKOS. Datasources: GKG, Wikimedia. Specifications: Shapes.

**Computation over graph structures (dataflow): layers class / occurrence orderings**

Ordering can be defined by the properties of their relations. Order defined by relations / occurrences and Role Kind subset / superset relation.

Graph: Node set, Edge set, Attributes, Global attribute. G : (u, V, E).

DGI: “every single node of the network is ‘mindful of the global structural properties of the graph’”. Representations.

Compute updated edge attributes.

Aggregate edge attributes per node.

Compute updated node attributes.

Aggregate edge attributes locally.

Aggregate node attributes locally.

Compute updated global attribute.

Three update functions (mappings).

Three aggregate functions (reductions).

Combinatorial generalization in graph networks.

**Encoding: Services Models, Backends, ML (Graphs, Embeddings, Autoencoders + Context)**

Resource occurrences: mode (tense / person / primitives: the, this, that, self, etc) "declination" (modal) metadata, inferred from contexts / aggregation / role kinds (role kinds role kinds specifications: from contexts / statements, negation, complement, equivalence, etc.). Ordering by properties of relations. Parameterized control flows / interpretations. Octal order encoding for flow control.

Property graphs.”The DOT Language “. N3 / Turtle rules.

Property dot graph encoding. Grammar: rules / productions. Specification (functional) DOM like notation / serialization / selectors (monads in specification runtime / components events / callbacks).

**Deployment: Node Context (domain peer)**

Node declaratively modelled by Resource hierarchy layers instances (deployment).

Holds metamodel Resource layers and IDs Graphs. Handles Template / Form encoded / inferred into / from graphs transforms (Activation)

Resource services (Naming, Registry, Index). Connectors: declarative contexts. Resource URL scheme for handling backends / services kinds I/O.

Distributed consistency

Higher level constructs (declarative language, APIs)

Use cases (layers "patterns": DOM / DCI / CDI / JAF)

Bindings with declarative metamodel description (layers "tags", templates, activation)

Application Language (component blueprints: layers). "Templates" from declarative metamodel descriptions

Applications: domains upper ontologies. Distributed consistency. Data / schema / behavior metamodel instances inferences / rules.

**Language. Runtime. Backend**

Browser / Endpoints consumption APIs (declarative / descriptive services / interfaces rendering)

RDF / OWL. Turtle / N3. XML / XSL. Java. Jena. Vertx. JavaScript. Lisp (metacircular interpreter)

**Deployment (Spring Framework)**

* Resolver: Identifiers, DIDs / Vert.x Message bus. Addressing / routing. Resource IDs: IDs Graph. Signatures (class / instance Resource IDs). Runtime definition.
* Runtime: Vert.x. CDI. Interfaces (register / inject Specification / Component). Interactions: Activation / Capability (Resolver / bindings).
* Specification: Protocols. Component(s) declarative model definition. Events / callbacks dataflow: executable Resource models descriptions. Context Role.
* Component: Specification instance, private state. Capabilities interface: Message IO. Context Actor.
* Activation (Template: input / Form: binding / Template: output). Capability (interaction) definition.
* Capability: Activation(s) instances (interactions). Available Message exchanges.
* Exchange: Message (data) blueprint.
* Message (Activation Capability exchange execution instance, encoded in DIDs). Interaction instance (data). Resource hierarchy transform.
* Application Graph: Streams / Flows (state): Contexts, use cases, available flows (Data / Context / Interaction DCI API.
* Invocation
* Aggregation : Activation
* Functional API / Service (Index, Naming, Registry) : Activation
* Resource
* Template
* Form
* Runtime: resolution / invocation application definition layout from source Resource layers metadata
* Context: (class / instance)
* Subject: (class / instance)
* Predicate (class / instance)
* Object (class / instance)
* Events / Callbacks
* Backend Specification: datasource / service IO.
* Spring contexts. Application, Runtime, Resolver, Specification, etc. (hierarchies, factories / builders). Routes (capabilities: Template / Form bindings). Classpath resources (JAR bundles).
* Runtime instantiates Specification Component(s). Component(s) interact by their capabilities (Runtime addressing / routing bindings: Resolver).
* Application Graph: Behavior of Resolver / Runtime, Specification / Component, Activation / Capability, Exchange / Message can be serialized / deserialized to / from class / instance quad statements.
* Given appropriate APIs (via Activation) one could invoke / execute specific behaviors (via CAM / learning / definitions events / callbacks) over Application Graph.

Resolution: Graphs (aggregation / layers) described by Activation(s) statements over input Resource stream (Ontology Matching).

Resolution : Activation

Context: (class / instance)

Subject: (class / instance)

Predicate (class / instance)

Object (class / instance)

Events / Callbacks

Resource Graph: Resource hierarchy layers. Message payloads.

Resource IDs Graph. Resolver.

Application Graph. Resolver.

Resource Graph: Metamodel backend encoding Application and Resource IDs graphs.