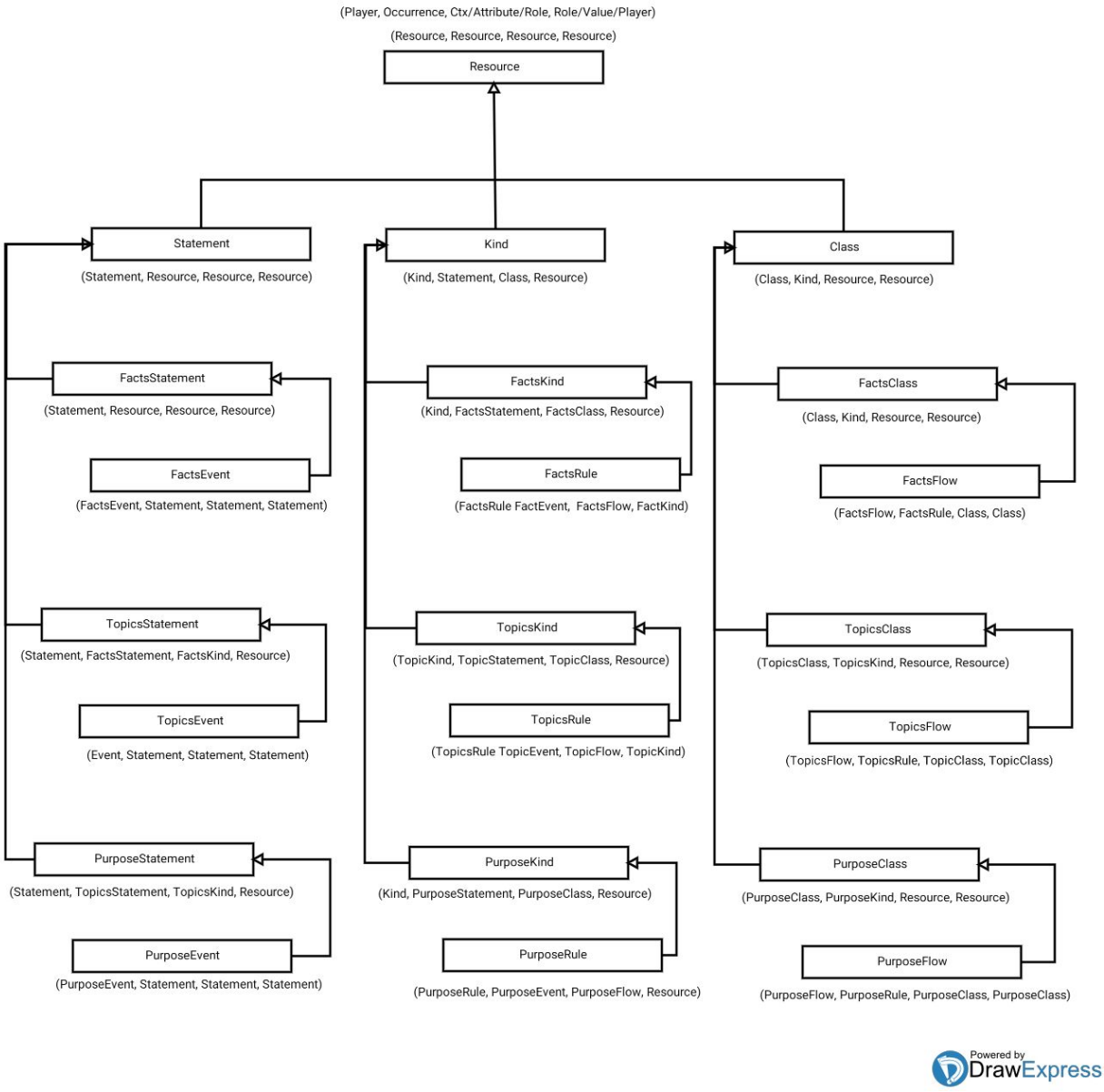
Introduction

Basically everything is modelled as RDF Quads. Classes in the diagram below (and Resources in general) have their instances represented as a Context (in the Quad) in which Resources have ‘occurrences’ or ‘parents’ (Quad’s Subjects) in which they play a role having ‘attributes’ (Quad’s Predicates) and ‘values’ (Quad’s Objects).



Resources, Statements and Kinds

Resources (SPOs) with an occurrence into an Statement have a Kind (type) corresponding to the Resource’s attribute and value (other SPOs of the Resource).

For example, a Subject ‘John Doe’ has a Subject Kind of ‘Employee’ in the Statement:

‘John Doe’, ‘worksAt’, ‘someCompany’.

Kinds aggregate Classes and may represent compound types (many attributes sharing their values.

Predicates and Objects are classified the same way than Subjects.

Model / Facades (for Containers)

Model : (Model, FactStatement, TopicStatement, PurposeStatement);

Aggregated Resource types (metamodel levels).

Profiles

Profile : (Profile, ?);

Templates

Template : (Template, ?); (Mapping).

Variables, expressions, wildcards. Patterns. Selectors.

Adapters / Ports

Adapter : (Adapter, Model, Profile, Model); (src, filter, dest).

Port : (Port, Adapter, Template, Adapter); (src, transform, dest).

[srcModelType][destModelType]Adapter class.

JenaModel; InMemoryModel; RDBMSModel

Events, Rules and Flows

Events: Order metadata / knowledge. Contextual registry functionality. Inference.

Event : (Event, Statement, Statement, Statement);

Rules: Identity, equivalence knowledge.Grammars. Alignment and merge inference. Patterns. Selectors. Naming resolution functionality.

Rule : (Rule, Event, Flow, Kind);

Flows: Attributes / links knowledge and inference. State graph resolution service (Index).

Flow : (Flow, Rule, Class, Class);

Example:

Event : (evt1, (anEmp, sal, lowSal), (anEmp, perf, goodPerf), (anEmp, sal, highSal));

Rule : (raiseSal, evt1, raiseSalFlow, empKind);

Flow : (raiseSalFlow, raiseSal, (lowSal, ‘salAttr’, 5000), (highSal, ‘salAttr’, 10000));

Containers

Container : (Container, Event, Rule, Flow); (data, context, interaction).

Messages

Algorithms. Message SPO: Data, Context, Interaction (Map, Filter, Reduce).

Encoding of ‘reactive’ expressions: event listeners / triggers. Routing / patterns.

Message : (Message, Container, Container, Container);

Subject Container apply(ied) to target Model (Map / actors).

Resulting Container is apply(ied) with Predicate Container (Filter / contexts).

Then, Object Container is applied to this result (Reduce / interaction). This interaction collects / returns into Object container, having ‘wildcards’ to be fulfilled from sender and then Message(d) back to resolve all patterns / selectors.

Resource matching:
Resource - Statement - Event.
Resource - Kind - Rule.
Resource - Class - Flow.

Map (align, rules) / Filter/Sort (events) / Reduce (links, attrs, classes).

Resources

Resource (Containers). Static factory / APIs.

Aggregators:

Resource's static class singleton (for each Resource type).

Abstract factory.

Parent / child of aggregated Resource types (metamodel levels).

Quad classes (C, S, P, O) parameterized (Java generics).

Tracks instances: hierarchical aggregated lists: (C(S(P(O)))).

Instantiation of children hierarchies.

Factory / CRUD / Functional methods.

Dispatch Message(s).

Query / browse (Classes / instances navigable graph, contexts).

Services implementation / facade.

Handle Model's backends (via Mapping specs).

Mappings (specs)

Mappings conforms the specifications to which a Model implementation relies in respect to its interaction with persistence or other IO mechanisms.

A Mapping is later leveraged by a Peer's services who handles the actual protocols and connections needed to realize it.

RESTMapping: Mapping which provides (given Peer's services) with the needed conceptualization to implement such interface:

URIs, Resources, Content Types, Representations, Verbs. HATEOAS Principles. DCI - JAF like. OData impl.

ObjectMapping: ORM like for RDF graphs. Clients (stubs, VM Activation).

Protocol (Container, Resources, Messages)

Container.apply(cont : Container) : Container;

Schema less protocol / storage. Dialog. Client sends Container and receives Container. Referer (context). ‘One method’ bi-di CRUD (metadata and semantics just in Container).

Mappings (other) implemented / interact with this abstraction (also schema-able / relational Mappings).

Model Backends

Mappings.

Peer services.

Deployment.

ETL / Dashboard

UX. Streaming CRUD. Planning (Rule, Flow, Event management. Process designer).

Data preparation / Refine. Reports, indicators. Document templates (forms, gestures, DAV). Design. Server. Client.

Interactions. Process (schema), Flows (instance) visualization. MDM, Governance, Traceability. BRMS, CEP. BPM. Workflow.

Alignment. Graph based schema merge / sync: Container protocol metadata. ISO / WebOWL Tools (export, endpoints).

Peers (services)

Binds Mappings / Models with specific protocols (persistence / communication mechanisms).

HTTP (REST, WebDAV), JMS, RDBMS (JDBC), JCR, SPARQL (RDF, OWL, ISO) services (interface implementation for each Model / Mapping type).

ISO Alignment

Align core model (Resource hierarchy classes) with an ISO OWL upper ontology. Backend metamodel.

Lab

Octal. Quad. Addressing encoding. URNs, Naming. Deep Learning. Cube. Algorithms facade. Functional Providers. Data addressable dataflow behaviors. Index, Registry. Clustering, Classification, Regression. Weka.

Order, Align: Opposite, inverse, complement.

Peer 'public html' folder (DAV / REST).

Naming (of Classes and Kinds). Resource type metadata (any SPO). Primitives (enumerable, operations) types. Naming (URNs) of Resources in context/occurrence with attributes and values. Concepts hierarchy / lattice.