

# 1 Introduction

## 1.1 Declarative application metamodels.

# 2 Architecture

## 2.1 Statement graphs. Dataflow (Reified CSPO Resource statements).

Resources (not only Statements) are 'reified' into a Resource object which takes into account Context, Subject, Predicate and Object as members of the reified Resource. (Statement, SPO or Kind).

```
Resource.holds(ctx : Resource, res :Resource);
```

```
Resource.apply(ctx : Resource, res :Resource);
```

## 2.2 Models and core classes: Declarations and instances.

Classes / interfaces. Factory, instantiation. Hierarchies. sameAs / domain / range / restrictions for RDFS/OWL representation.

### **Declarations:**

```
Resource : Class (Predicate, Resource, Resource, Resource);
```

```
Predicate : Resource (Set, Resource, Resource, Resource);
```

```
Set : Predicate (Statement, Predicate, Resource, Resource);
```

```
Statement : Set (Model, Set, Predicate, Resource);
```

```
Model : Statement (Container, Statement, Set, Predicate);
```

```
Container : Model (Adapter / Port, Model, Statement, Set);
```

```
Adapter / Port (Runtime, Container, Model, Statement);
```

### **Instances:**

ResourceInstance : Instance (PredicateInstance, ResourceInstance, ResourceInstance, ResourceInstance);

PredicateInstance : ResourceInstance (SetInstance, ResourceInstance, ResourceInstance, ResourceInstance);

SetInstance : PredicateInstance (Statement, PredicateInstance, ResourceInstance, ResourceInstance);

StatementInstance : SetInstance (ModelInstance, SetInstance, PredicateInstance, ResourceInstance);

ModelInstance : StatementInstance (ContainerInstance, StatementInstance, SetInstance, PredicateInstance);

### **Statements:**

(Class, Instance) (Class, Instance) (Class, Instance);

## 2.3 CSPOS, Kinds and Statements. Reification. Models.

Contexts, Subjects, Predicates and Objects declarations and instances are direct subclasses of Resource and ResourceInstance respectively.

A special type of Resource are 'Kind's. Each SPO in a Statement has its corresponding Kind which has an 'attribute' and a 'value' in respect to its position in the triple. For example, the triple:

(Peter) (worksAt) (IBM)

Has a 'SubjectKind' of (worksAt, IBM) for its Subject (Peter). Subject's attribute and value are (worksAt) and (IBM) respectively. This metadata could be used for basic type inference by aggregating Employees (worksAt domain) and specific employees (which work at IBM).

The same holds for classifying Predicates and Objects into their types and their corresponding meta-types. Different Model layers (discussed later) have its own (Predicate based) SPO Sets definitions and, thus, their own Statement and Kind structures.

## 2.4 Application core Models (layers) class and instance declarations.

The first Model (Facts) is built from raw input triples (SPOs) and its Kinds are aggregated according its Statements and Resource occurrences.

Then, TopicModel's Statement and SPO instances are Resource Sets built from Predicates which take previous Model Sets (Facts) Statements, Kinds and SPOs for building Model's SPO respectively. The same pattern is applied for building BehaviorModel layer.

### **FactModel**

SPO: (Subject, Predicate, Object).

Kinds: SubjectKind (PO), PredicateKind (SO), ObjectKind (SP).

Statements: (FactCtxPred, Subject, Predicate, Object).

### **TopicModel**

SPO: (Object: FactModel's Statements, Concept: FactModel's Kinds, Sign:FactModel's SPOs).

Kinds: ObjectKind, ConceptKind, SignKind.

Statements: (TopicCtxPred, Object, Concept, Sign).

### **BehaviorModel**

SPO: (Topic: TopicModel's Statements, Scenario: TopicModel's Kinds, Player: TopicModel's SPOs).

Kinds: TopicKind, ScenarioKind, PlayerKind.

Statements: (BehaviorCtxPred, Topic, Scenario, Player).

### **ActionModel**

SPO: (Event : FactStatement, Flow : TopicStatement, Rule : BehaviorStatement).

Kinds: EventKind, FlowKind, RuleKind.

Statements: (ActionCtxPred, Event, Flow, Rule).

### **Grammars (Facts example)**

SPO: (SubjectKind, PredicateKind, ObjectKind) non-terminals.

Kinds: Subject, Predicate, Object (terminals).

Statements: Kinds / SPOs (rules, productions).

### **Services**

## 2.5 Services. Reactive Models for Containers.

Services are a special type of Models held by Containers in their scope which take into account special types of events / messages. Services listen to / send messages which keep models in sync with metadata they process, transform and provide.

Materialized service commands statements.

```
Resource.apply(ctx, cmd);
```

## 2.6 Resources reference model (Index, Naming and Registry Services).

Reference model are the means to provide an unified API over Resources to query / manipulate its state via the notion of 'mappings' (links and attributes) held by an Index Service, a hierarchical location in a context graph handled by a Registry Service and a name enabled order comparison mechanism provided by a Naming Service.

Services are Models and thus reactive to its Container Model's events and push events to its Containers. Services apply to 'schema' (declarations) as to instances and statements.

## 2.7 Functional API

Functional API is the core low-level client API exposed by the Runtime hierarchy for interaction with its held Resources. Most of the features rely on Resource's methods 'holds(ctx, res)' and 'apply(ctx, res)'.

### 2.7.1 Dialog / Dataflow Protocol. Message Statements

Statements (messages) are produced / consumed by Models. Reactive dataflow is performed 'applying' 'held' resources (materializing results from this operations).

Messages IO performs a dialog like protocol in which wildcard and variable placeholders are allowed. Message submission may resolve into a response which has placeholders to be filled which produces client (original sender) to receive the message for trying to resolve missing data from its own models.

### 2.7.2 Containers, Models and Messages discovery and binding (Profiles)

One should be allowed to interact with a Model (Facade) Container querying for its allowed message / events (Statements) and retrieving the Sets (Profiles) for which the messages holds (Predicate / Template).

As with the message dispatch approach (below) this 'discovery' may be performed inside-out (from Resource / Template) using routing tables or iterating from container through children.

This way querying, for example, Action layer facade should bring declaratively (through reference model browse) which classes, instances, contexts, interactions and roles are available for a given domain.

### 2.7.3 Client binding: DOM / DCI patterns over declarative Action Model layer metadata

From declarative layers metadata build DOM (Dynamic Object Model) with classes, instances, contexts, roles and interactions.

JAF enabled content types and commands.

Monads. Materialize messages / dialogs. Addressable contexts / interactions.

## 3 Features

### 3.1 Align, merge. Identity resolution (by reference model). Algorithms.

'Resource.holds(ctx : Resource, res : Resource)' is the main method for alignment, merge and identity (equivalence) resolution. It relies on Resource's reference model Index Service.

Index Service listen to events over Model Resources and resolves its mappings in context (attributes, links, relationships). It also posts events materializing this knowledge into statements. For this it uses aggregation, discovery, learning, classification and regressions over previous knowledge.

### 3.2 Attribute and links discovery (by reference model)

Index Service.

Resource.apply(ctx : Resource, res : Resource);

### 3.3 Ordering and temporal alignment (by reference model)

Naming Service listen to events over Model Resources and resolves its positions in contexts (prev / next). It also posts events materializing this knowledge into statements. For this it uses aggregation, discovery, learning, classification and regressions over previous knowledge.

### 3.8 Type inference (by reference model)

Index Service.

Attribute and links discovery.

### 3.5 Relationships browse

Registry Service listen to events over Model Resources and resolves its hierarchies in contexts (parents / children). It also posts events materializing this knowledge into statements. For this it uses aggregation, discovery, learning, classification and regressions over previous knowledge.

Example: (Peter, Joe) : where neighbors, then friends, and then partners. Transitions. Truth values (temporal, for ordered 'names').

### 3.6 Grammars

#### **Grammars (Facts example)**

SPO: (SubjectKind, PredicateKind, ObjectKind) non-terminals.

Kinds: Subject, Predicate, Object (terminals).

Statements: Kinds / SPOs (rules, productions).

Primitive terms.

Opposite terms.

Negation terms.

Inverse terms.

Complement.

Materialize terms (context, role, term).

### 3.7 Datatypes

Primitive (enumerable) types.

Grammars. Rules (class). Production (instance).

## 3.8 Dimensions

Dimension. Unit. Value.

Materialize: (someValue, someProp, someObj).

## 3.9 Learning

### **Service Models Features**

Aggregation.

Classification.

Regression.

Dimensional.

Alignement.

FCA. Augmentation.

## 3.10 Upper ontology alignment. RDFS/OWL Model

Event propagation keeps RDFS/OWL Jena model (upper ontology) aligned and in sync with core models. RDF / SPARQL endpoints.

## 4 Backend

Learning and Big Data features.

### 4.1 Jena backend: Reified statements. RDFS / OWL

All declaration, instances and statements conform to a core ontology and are represented into a Jena model.

### 4.2 WebDAV / JCR (Registry)

Hierarchical datastore implementation for Registry Service. Contextual: a Resource may have multiple occurrences in different contexts.

## 4.3 Lucene / Solr (Index)

Indexing component for Resources (attributes, links, relations). Contextual. Same resource multiple occurrences. Index Service backend.

## 4.5 JNDI (Naming)

Namespace domain resolution for ordered set (flow) of Resource (states) in context. Naming Service backend.

# 5 Deployment

## 5.1 Runtime

Base environment configuration for core models execution / interactions.

## 5.2 Peers (Container)

Main Runtime deployable / bindable unit into Clients / Servers.

## 5.3 Clients (Ports)

Exposes its underlying Peer as a service (API).

## 5.4 Servers (Adapters)

Configured to be bound as a source for containers (datasource).

# 6 Applications

Dashboard: peer manager console.

Declarative (model driven) architecture through language bindings of Action layers Facades, DOM (dynamic object models), JAF (JavaBeans Activation Framework) and DCI/MVC design pattern over Protocol/Dialogs.

## **Runtime**

Environment configuration.



Adapters / Ports Containers set up (datasources, peer application Models bindings).

Protocol endpoint. REST RDF Service endpoints. Queues, routes, transformations. Dispatch.

Messaging / Events dispatch:

1. Adapter / Port sends / receives Resource in payload.
2. Container iterates its Models
3. Model iterates its Statements
4. Statement iterates its Sets
5. Set iterates its Predicates
6. Predicate iterates its Resources
7. Retrieve / build Statement. Return (eventually incomplete, vars / wildcards) Statement

For reverse lookup (from Resource) build resolution / routing tables. Dataflow (reactive) traversal may start from leaves (Resource Object, Predicates, Subjects till Contexts) using tables or from outer Resources, each case being Resources visited (holds/apply invoked on them) and using their response for building adequate objects in the hierarchy.

Routing tables lazily populated on demand: when no data is contained for input of a particular Resource into routes tables then the full traversal is performed populating the tables accordingly for later use.

Resource.apply(Resource) : Predicate

Resource.apply(Predicate) : Set

Resource.apply(Set) : Statement

Resource.apply(Statement) : Model

## Lab

Octal order relation encoding.

Encoding and addressing of IDs. Mappings to URIs.

Patterns: ie.: SK matches Subjects. ID ops.

Statements materialization: Triple IDs (x, y) (x, y) (x, y). Classes / Instances IDs. Each with its context, occurrence, attribute and value (CSPO).

## Learning

Reference Model inference: Generate reference model services messages / events from materialized statements. Functional mappings inference: Index Service events (ID equivalence resolution). Registry Service events (Hierarchical aggregation). Naming Service events (Alignment and contextual ordering/sort).