Business domain translation of problem spaces

June 2017. Sebastián Samaruga (ssamarug@gmail.com)

Abstract: The concept revolves around a network of (perhaps existing network's) Peers which interacts in the aims of fulfilling business Objectives (Task flows) for which Peers are selected to perform regarding their Profiles and a given Purpose they and Objectives are proposed to accomplish. A Purpose is a kind of 'abstract goal' to be met. Profiles are the (maybe evolving) capabilities for the resolution of Objectives problem kinds.

Given business domains 'problem spaces' a 'translation' should be made which encompasses given a set of problem Objectives to be solved in one domain (maybe because of Events in that domain) another set of co-requirements in other domains which triggers new Objectives into the flow which shall be accomplished for the global Purpose to be met.

An example: In the healthcare domain an Event: flu diagnose growth above normal limits is to be translated in the financial domain (maybe of a government or institution) as an increase of money amount dedicated to flu prevention or treatment. In the media or advertisement domain the objectives of informing population about prevention may be raised. And in the technology sector the tasks of analyzing and summarizing statistical data for better campaigns must be done.

The technology empowering such endeavor must not be other than that of the graph-oriented featured semantic web paradigm, with tweaks into functional programming concepts and Big Data / Machine Learning inference providers.

Application Protocols:

For most use cases applications needs to be deployed in ways where they are allowed to interact with other applications and produce or consume services.

Although the proposed application mentioned in this document is thought to be implemented in an end to end full stack manner (from presentation through business logic to persistence) the core components meant to be used here are distributed and functional in nature.

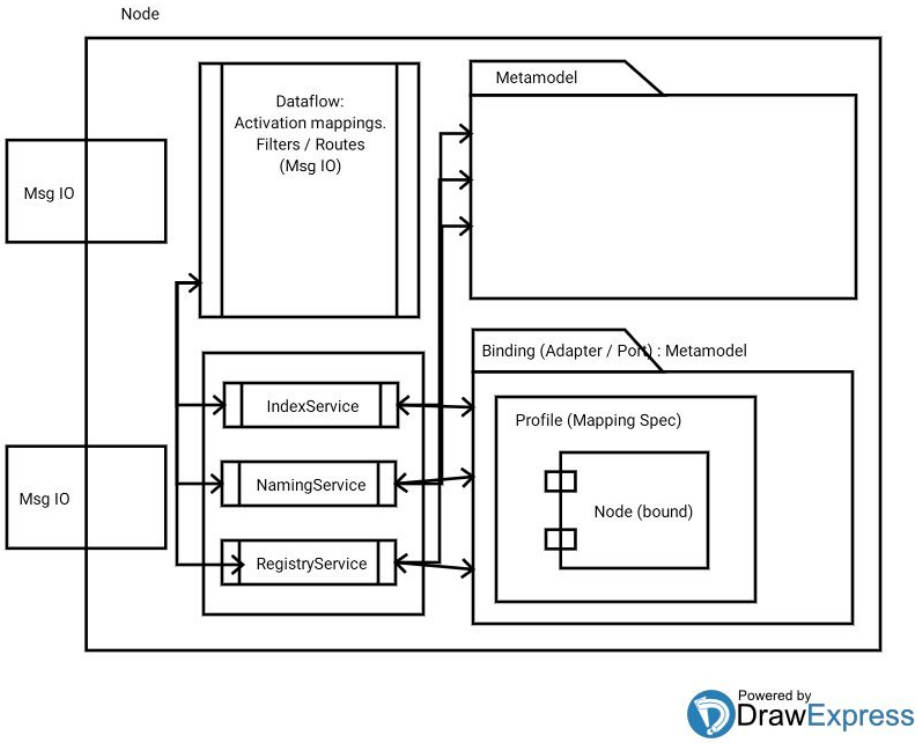
Implementations of the framework shall allow for the discovery and publishing of profile driven endpoints.

Traditional persistence mechanisms synchronization and standards based protocol bindings (REST, SOAP, others) enables for integration. And a custom schema less dataflow (RDF based, dialog scoped) protocol is meant to have API bindings in different platforms for application development.

Deployment Peers (Adapter / Port Nodes):

The core architecture deployment is based on Containers of Peer Bindings which allows for Node endpoints Profiles to be discovered / interconnected via the protocols specified by their Mappings / Specs.

This allows for direct (custom semantic dialogs / protocol) Peers integration as also the direct integration of traditional application backends with synchronization capabilities.



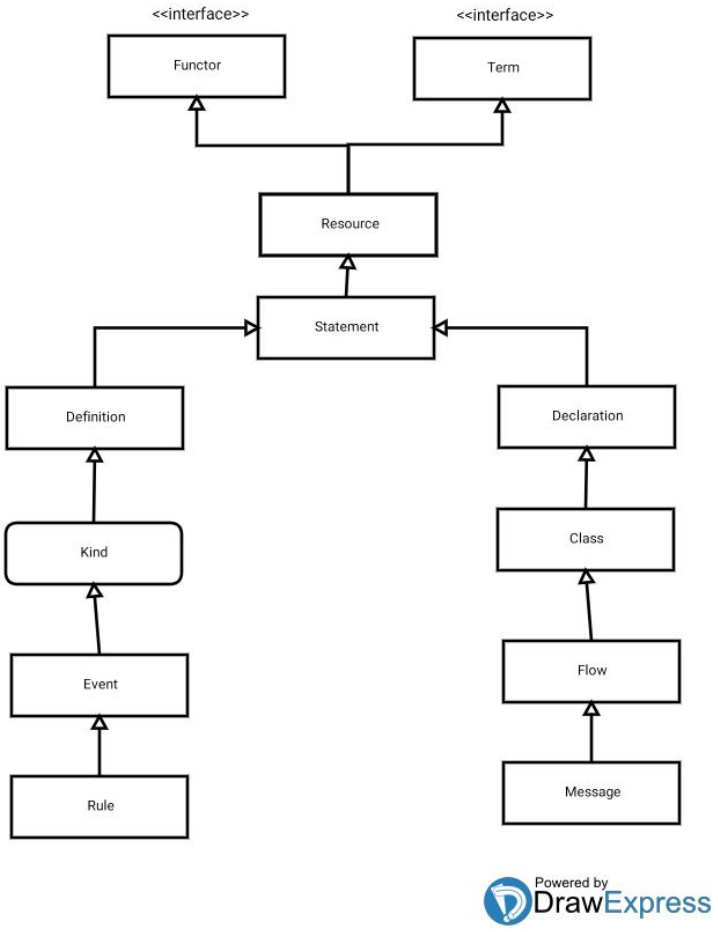
Deployment

Node
Metamodel
Binding : Metamodel (Adapter / Port of Profile wrapped Nodes: Datasources, Peer Nodes, Protocol endpoints, ML / Big Data engines).
Activation: state mappings : routing / filter.
Services:
IndexService : Facts (Statements, Alignment).
NamingService : Topics (Events, Link / attribute resolution).
RegistryService : Behavior (Messages, Order resolution).
Facades (not shown): wraps Node via Templates (Profiles). Higher level APIs (domain specific).

Profile Mappings / Specs: provides anything to - from RDF semantics (schema less). Example specs (Nodes): RESTEndpointSpec, SOAPEndpointSpec, SPARQLEndpointSpec, RDBMSEndpointSpec, GUISpec (via Facades).

(Embeeded) Node Bindings may contain Profiles of Bindings that should be met by container Nodes (feature injection).

Resource Metamodel:



All hierarchy classes (including Functor and Term interfaces) are defined in terms of RDF Quads. Each Quad instance context aggregates other Quads instance contexts according their meaning.

Later we'll see how aggregation combines with dataflow activation for the query/traversal and inference over the resources graph.

Data input (Statements) activates (dataflow) on contexts (Definitions) and interactions (Declarations). Node infers / aligns / augments from input Statements. Definition / Declaration hierarchy: AST / Monadic parser combinators on input Statements. Node aggregation of Kinds as Statements (features).

Resource is of the form:
(Player : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);

Definition Resource occurrences:
Kind: Statement.
Event: Kind?
Rule: Event?

Declaration Resource occurrences:
Class: Kind?
Flow: Event?
Message: Rule?

The Quad statements are of the form:

Quad : (Player, Occurrence, Attribute, Value);

Functor (functional wrapper of Resource):
(Functor, Resource, Term, Functor);

Term (bound function / dataflow op):
(Term, Resource, Functor, Functor);

Quad patterns:

Resource : (Resource, Resource, Resource, Resource);

Statement : (Statement, Resource, Resource, Resource);

Definition : (Definition, Resource, Declaration, Definition);

Declaration : (Declaration, Resource, Definition : lhs, Definition : rhs);

Kind : (Kind, Resource, Class, Kind);

Class : (Class, Resource, Kind, Kind);

Event : (Event, Resource, Flow, Event);

Flow : (Flow, Resource, Event, Event);

Rule : (Rule, Resource, Message, Rule);

Message : (Message, Resource, Rule, Rule);

Statements further aggregates:
Facts : Actual input Statements.
Topics : (Topic, Fact Statements Fact Kinds, Fact Resources);
Purposes : (Purpose, Topic Statements, Topic Kinds, Topic Resources);

Resource Functor: Resource<T> hierarchy. Resource hier : AST. Parser Functors. Bound functions (Terms):

Example:

('Someone' : Resource).flatMap(Employee : Kind) : EmploymentKind : Someone's jobs.

Kind : (Person : Kind, Someone : Resource, Employee : Class, Employee : Kind);

Class : (Employee : Class, SomeoneEmployee : Resource, Employee : Kind, SomeoneEmploymentsKind : Kind);

Features (functional programming / providers):

Parsing: Aggregation of basic inputs (Resources, Statements) inferring their Kinds and Classes (a Kind is a Resource Class occurring in an Statement context) allows for further arrangements leading to infer / learn new knowledge. Resource classes are (monadic) parsers of their own types (consumes their occurrences in quads) and dataflow activation allows for declarative composition of inputs into new knowledge.

The comparison of resources in respect to a given Term (parent, child, previous, next, current, first, last, single) predicate in a given context (Term, Resource, Context, Resource) is meant to allow query based inference algorithms and also the (also dataflow enabled) integration of external machine learning / Big Data engines such as Google TensorFlow (given the correct encoding of resource IDs).

Metaclass - Class - Instance relationships expressed also as order comparison relations (example: a subclass has a superset of the attributes of its superclass, Resources defines other Resources by their aggregation contexts).

Data layer: Represented by Statement, Event, Message hierarchy.

Context / schema layer: represented by Definition, Kind, Rule hierarchy.

Interaction / behavior layer: represented by Declaration, Class, Flow hierarchy.

Order augmentation: Parse Flow, Rule, Message orthogonal aggregation. Parent attribute sets (Declaration) in Definitions / Statements determines order.

Alignment augmentation: Parse (orthogonal) of Declaration, Definition, Statement to compare ‘abstract shapes’ of aggregated graphs.

Attribute / Link augmentation: Parse orthogonal Event, Kind, Class ‘attribute sets’. Sets rels determines merge.

Resource API

Activation Dataflow

Variables. Placeholders

Primitives

Dataflow Activation:

Messages protocol.

Metamodel Resources activate on C-S-P-O.

Active mappings (inputs / outputs activation graph): filter / routes.

Given a Resource, for example a Message which aggregates three Model(s) that could be: a Facts Model, a Topics Model and a Purposes Model, interaction via dataflow could be a certain Fact (input data Statement) becoming ‘enabled’ (pattern) which ‘enables’ / activates the Message for consumption of the next (info / Topic) Model which then triggers the Purposes Model pattern ‘active’ in Message context.

Triggering and activation of context inputs and outputs (Statements SPOs) is event driven (dataflow) and Resources (Terms) which aggregates its activation states acts as arguments and results of the ‘operations’ defined by their contexts.

Functional features (Functors / Terms)

Parsing / Traversal / Transforms (fmap)

Graph encoding. Naming scheme (Profile IO translation)

Application Stack:

Business Applications (Social Peers Interactions).

Dashboard: ETL / Refine front end.

Application Clients (Platform bindings of Protocol semantics). Functional API (DCI).

BI / Big Data (engines / data sources).

MDM / Governance / ESB / Workflow / BRMS.

Features

- Reify CSPOs as Resource (Statement) example: (S : player, Statement : occurrence, P : attribute, O : value).
- Substantive, Verb, Adjective (computer, computes, computed). Statement (substantive by extension), Definition, Declaration. Order align (verb states). Infer substantive, verb, adjective from (reified) kinds: new statements, definitions, declarations. Learn Events, Kinds, Class, Message, Rule, Flows. (State, Action, Passion).
- Equivalent properties by equivalent property domain / range (kinds). Domain / range kind alignment, links, order inference by equivalent property kinds. Encode RDFS Statements.
- OWL: properties / taxonomy, OWL restrictions: Encode (upper ontology, alignment of inferred facts). Definition / Declaration (comparisons: schema, links, ordering) based alignment.
- OWLNode: Reasoner, transforms / templates (IO Facades node ontologies). Metamodel Bindings / Profiles.
- Metamodel: Jena / RDFS Reasoner. Functional API. Dataflow. Services (IO) activation.
- Parameterized type M<T>.
- Unit function (T -> M<T>).
- Bind function: M<T> bind T -> M<U> = M<U> (map / flatMap: bind & bind function argument returns a monad, map implemented on top of flatMap).
- Join: liftM2(list1, list2, function).
- Filter: Predicate.
- Sequence: Monad<Iterable<T>> sequence(Iterable<Monad<T>> monads).
- Order by contextual comparator of Definition Declaration.