# Business domain translation of problem spaces

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

Abstract. Keywords. Introduction (business domains integration use cases).

Streamline and augment analysis and knowledge discovery into enhanced declarative and reactive event driven process flows.

**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.

Besides some initial type inference over RDF datasets the features here discussed not only focus on ontology engineering or modeling. The aim is more like to provide an engine (some like DOM / JavaScript is for HTML) for whatever RDF can be provided for it from whatever sources, not only DBpedia but seamless integrating RDBMSs / REST / LDP or any other backends / datasources / protocols.

An important goal will be to be able to work with any given datasources / schemas / ontologies without the need of having a previous knowledge of them or their structures to be able to work with them via some of the following features:

Achieve plain RDF serialization to-from any service / backend through Node Bindings (below).

Model statements into layers, each one having its own 'abstractions' for the roles each statement parts play:

Data layer: Infer type information, Classes / Kinds roles. Properties (domain / range role Kinds). From facts layer statements (data) infer which property kinds holds for which domain / range role kinds:

Who is a Person.
Who is a Business.
Which relations (properties) holds between them.

Information (schema) layer: Events which entail Flows given certain statements (relations) roles.

aPerson (hired), worksFor, someCompany (hires);

Knowledge (behavior) layer: Given certain Rules some Messages are ‘activated’.

betterJobOffering, employeeBehavior, jobSwitching;

Knowledge, information and data behavior (CRUD) entails ‘propagation’ into upper / lower levels of statements updating them accordingly via dataflow graphs (below).

Data example: aProduct, price, 10;

Information example: aProductPrice, percentVariation, +10;

Knowledge example: aProductPriceVariation, tendencyLastMonth, rise;

Features should be provided by the framework / datastore (alignment):

Align identity: merge equivalent entities (with different URIs). Align types (schema / promotion: infer type due to role in relation).

Align attributes / links: augment knowledge (properties and values) about entities (because of type / role alignment).

Align ordering: sort entities regarding some context / axis (temporal, causal, composition and

other relations).

Implement functional query / transformation language: for a given entity / concept (Monad) being able to browse / apply a function which entails some other result entity (Monads).

Resource endpoints. RESTFul interfaces for metamodels interaction. Provides / consumes (feeds / streams) events / messages declared via dataflow engine. The datastore should be a HATEOAS web application.

Implement XSL Driven declarative dataflow engine: Streams ‘pipes’ declaratively stated in XSL for reactive behavior definitions. Functional query / transform semantics. Datastore application publish / subscribe to this (request / response 'filters').

Implement a 'Node' abstraction: Integration of diverse (wrapped into metamodels) datasources / backends / services / protocols via the implementation of service contracts and exposing a plain RDF IO (dialog protocol) interface. A Node Binding to other Nodes shall allow to integrate via dataflow semantics diverse datasources / datastores.

Provide discovery services for data / schema / behavior (bus / stream events actors and roles). Bind Profiles by ontology metamodel alignment: Registry: Ordering alignment, Naming: ID and instance matching alignment, Index: Links and attribute alignment. Big Data and EAI integration patterns. Reactive adaptive dataflow containers (criteria instead of hardcoded addresses).

**Topics:**

**Nodes:** Server Peers. RDF statements IO / ETL syndication / sync to - from any backend / datasource / service / protocol. Bindings: Engine Peers wrap exposed protocols. Schema aware ‘less’.

Services: Nodes with IO (registry, index, naming, ML, Big Data, reasoner, BRMS, etc) provider Nodes.

Discovery by profile: criteria / patterns instead of endpoint addresses. Referrer contexts. DCI. Bus: dataflow streams (pipes) actor / role pattern.

**Layering:** Aggregation. Data, information, knowledge. Resource hierachy meta models. Quads (reified). Graph serialization (Apache Jena). Entities: Type (Class / Kind), Data (Event / Flow), Behavior (Rule / Message). Type (class / metaclass) inference.

**Alignment:** Behavior: Order (registry, JCR provider). Schema plus roles: Rels / links (index, Apache Lucene provider). Data:Identity (naming, JAF / NLP provider).

Discovery. Internal upper ontology. Sust, Verb, Adj. Reify layers entities. Terms (reified grammars, dimensional, rules). Constraints. Shapes. Domain / range, ordering rels comparisons inference.

**Functional (query / transforms) API:** Monadic. Reactive dataflow (distributed nodes activation graph). IO format: Resource metamodel hierarchy.

Terms: internal upper ontology aligned. Variables, placeholders. Pointers: resolve subject in role in context (Semantic pattern IDs).

**XSL Declarative reactive streams dataflow:** DCI (Message, Subscriptions, Processor). Functional transforms declarations plus Terms. Events pipes / routing. Resource ‘activation’: layers / reified Nodes propagation. Actor / role. Reified Resources into transforms (dynamic / patterns) addressable. DCI dynamic behavior.

**RESTFul Resource hierarchy endpoints:** XSL filters / activate inputs and outputs. Generates derived (activated / augmented) statements. Activates functional layers aggregation. HATEOAS: JSONLD / HAL.

**Engine:** Generic client (Peer, browser). Transforms (wraps) Node into protocols / schemas. ‘Semantic ORM’ (Dynamic Object Model, Actor / Role DCI, JAF). HTML DOM / JS like platform bindings. Declarative discovery / activation. Reactive platform bindings (Java, JS, etc) over RESTFul Resource endpoints.

**Business Domain Translation of Problem Spaces:** TBD.

**Generic client (browser):** Peer configuration, discovery, ETL. Dashboard (default app: CRUD / Social / Process workflows, etc.).

TBD.

**Architecture**

Backend Nodes (features): Layers (types, aggregation), Alignment (id merge, order, links / rels), Functional query / transform (as Resources), XSL declarative pipes, REST Resource endpoints (pipes / events / message aware: methods). Each Node declaratively listens / augments other Nodes.

Nodes: Bus (syndication) Message driven activation / dataflow (pluggable discoverable features: declarative 'Nodes' activate on inputs, augments meta models, bus IO). Bindings: nodes implementing backends / protocols / datasources IO (sync).

Browser: Client Node (over protocol Node). Engine Node API: features MVC / DCI DOM / Functonal REST metamodel. Engine Node declaratively activates on metamodels (Node IO). Renderers (UI / protocol Bindings) implements Node. Engine: local stub.

Dashboard. Admin. Features (integration). Domain translation. Alignement (merge) Actionable items (dataflow, activation, drill, mine, browse). Stubs other features.

Resource URI scheme / format for proper feature / activation patterns. Naming, index, registry features (used by alignement features pipes). Express features as XSL declarative templates? (Feature / Node object model / interfaces).