Products And Services Community Exchange Network:

Going through my most recent attempts of having something concrete for sharing in plain English I realize one mistake I'm committing: I'm trying to describe combustion vehicles (Hypermedia Applications) saying that petroleum exists (Semantic Intelligence).  
  
As long as my post are going I've just got a stack of (incoherent) "analysis" documents as the result of my work. And I had only those until now because I was stuck because of the previously mentioned mistake (ah, and because of my Bipolar Disease maniac episodes...).  
  
I should try to describe applications instead and see how and where fuel should burn properly inside a motion vehicle to generate traction. Every semicolon I write is updated into my GitHub repository, so, sorry if you browse that "scrapbook" and you don't find anything even intelligible.  
  
First, I'll try to describe a "problem" (problem "spaces" in this case) and how a Purpose driven user Community achieves its Goal(s) by means of Goods, Products and Needs satisfaction (ontology levels: from abstract upper ontology to user gesture command in user interface / service invocation).  
  
The problem is to organize interdisciplinary (multiple domains) Task(s) in a Purpose fulfilment network with Actors, Contexts and Roles (with attributes and values). Problem spaces (domains) are declaratively stated by DCI[1] design pattern: Data / Context / Interaction use cases definitions and instances.  
  
Collaborative Federated Actor network complying determinate Profile(s) satisfying specific Product / Good / Need abstraction playing determinate Role in use cases Context.  
  
Domain Translation between business domains, example: orders, delivery, invoicing (micro) services Model instances are the means by which distributed disparate data, schema and behavior of different sources (applications, services) integration could be performed by means of Semantic Intelligence and Augmentation Protocol(s).  
  
A domain can be defined in terms of a set of actions / tasks with the Purpose of satisfying some Goal solving the Need for a Good producing / gathering a Product. Ontology. Purpose as Goal “class”.

The principal focus is to deploy a (social) Collaborative peer (Actor) network for which entities and individuals develop Profile(s) which acquaint them with Purpose resolution capabilities. Then, according peer’s specific needs (domain Goals) the application orchestrates interactions needed for Product(s) Task(s) accomplishment.

Ontology:

Domain / Actor / Context / Role / Product / Good / Need / Purpose / Task / Goal / Exchange.

Domains: data, schema and behavior of business applications (ERP, CRM, BI, SCM, HMS, etc.).

General purpose business domains problem resolution / tasks, goals accomplishment helper tools.

Syndication (contextual hypermedia activation): QA. Polls. Learning. Profiles. Guided task (wizards), guided editors: Context: Goal / Purpose.

Contents: Wiki view of augmented knowledge. Addressing. Hypermedia. API (Wiki) render nodes / links semantically browseable.

Backend: Nodes / Protocol.  
  
SoLiD:

<https://solid.mit.edu>

DIDs (Blockchain dApps):

<https://w3c-ccg.github.io/did-spec/>

<https://ont.io/#/>

Executable models (flows): testing results, prompts, scoring.

Applications (use / implement like):

Drive / Jira / Trello / Keep / Mural / Tasks / Calendar.  
  
Ontology levels abstractions (data, schema, behavior): service / user interface rendering (activation).   
  
Dashboards components (widgets / media / extended content types / addressing).  
  
Hypermedia Activation. Addressing. Link extended content types resources elements / parts with other resources addressed elements.

Objectives:

Develop Protocol (APIs) to facilitate Enterprise Application Integration (EAI) by means of Semantic technologies and Machine Learning. Ontology matching driven data, schema, behavior inference / aggregation / matching. Reasoning and learning over different consolidated backends alignments.  
  
Distributed P2P (Blockchain) approach of data synchronization between peers for ease of deployment patterns election and datasources integration (APIs, microservices, etc.).  
  
Data alignment:  
  
Determine if two instances (example: records) of two different backends or services refer to the same entity (Customers : John D. / Employees : John Doe).  
  
Schema alignment:  
  
Determine, for example, meaning and equivalences between diverse (aggregated / composite) schemas (equivalent classes, equivalent attributes, equivalent roles).  
  
Behavior alignment:  
  
Determine meaning and equivalences between (aggregated / composite) behavior contexts and behavior contexts invocations / interactions (Appointment / Interview, anAppointment / anInterview. Behavior flows aggregated from backends / services learning).

Alignments Augmentations:  
  
Activation: type inference : classification (determine class / metaclass / roles for entity attributes and values).  
  
Activation infer attributes / relations : clustering (from multiple occurrences of same entity in diverse data sources).  
  
Aggregation: infer roles in contexts: regression (Person class in Employment interaction : Developer role).  
  
Integration of addressable resources. Reactive I/O (sync back ends). Content type driven semantic augmentation / annotations.

Augmentation of distributed resources. Annotations (Semantic / ML). API for resource / schema / interactions exploration / protocol for message based API "dialogs" execution. HAL (Hypertext Application Language), OData (REST) like interfaces.  
  
Example: Google Drive / Google Knowledge Graph APIs Augmented with ML / Semantic intelligence tailored for specific domains / application kinds.

Augmentation. Ontology matching. Hypermedia augmentation protocol. Browser / Client APIs.

URIs API for annotating network retrieveable resources metadata. Content type / model driven augmentations / activations (models features / outputs). Subject attributes / values. Occurrences contexts / roles. Paths, pointers, locators. Example: annotate document URIs (parts, sections, mentions), annotate images URI (whole image description, coords: classes, individuals), annotate DB, table, row, column, value URIs, annotate / describe service / APIs URIs. Hypermedia protocol composable with other (described / annotated) APIs / resources. Example: Drive APIs.

What my attempts are about where, in the beginning, to match different URIs or identifiers which refer to the same entity (in different databases / ontologies, for example) to perform some kind of "ontology matching".  
  
Then I've tried to develop a mechanism for using RDF Quads for encoding an object graph (and a layers class hierarchy) using Contexts to denote the class of an instance, Subjects to denote class instances and attributes (members) and values: Predicates / Objects.  
  
Quads are "reified" as Resource(s). Also, Resource is a functional wrapper reactive and event driven of an URI. And an URI could be implemented with whatever backend which could produce or consume events (databases, services, etc.). Resource layers hierarchy (Context) is to be implemented by an actor / role type object pattern.  
  
Then I've realized that some basic type inference could be performed with, for example, aggregating Subjects with the same predicates (Subject Kinds). Idem for Predicates, Objects and Contexts. I've also realized that plain "facts" statements could be aggregated in the previously mentioned class hierarchy to abstract further, from plain data, instance / class layers of what I call data / schema / behavior layers. Higher layers (i.e.: Behavior) "aggregate" lower layers.  
  
Layers shape is as follow:  
Resource : Functional URI wrapper.  
(Context : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);  
  
Each layer abstract:  
  
Statement (data instance):   
(Statement, Occurrence, Attribute, Value);  
someOne buys someProduct  
  
Entity (data class):  
(Entity, Statement, Occurrence, Attribute);  
someBuyer, someProduct (Entity);  
  
Role (schema instance):  
(Role, Entity, Statement, Occurrence);  
Buyer, Product (Role);  
  
Class (schema class):  
(Class, Role, Entity, Statement);  
Person, Good (Class);  
  
Flow (behavior instance):  
(Flow, Class, Role, Entity);  
someBought (Flow);  
  
Behavior (behavior class):  
(Behavior, Flow, Class, Role);  
Buy (Behavior);  
  
This "aggregations" are part of what I call "Augmentation(s)": Aggregation, Alignment and Activation are ones of those, which are functional transforms described declaratively in an object graph metamodel. The act of applying an Augmentation implies one source Resource (context), one template Resource (transform) and a resulting (set of) Resource(s).  
  
One also could Augment Resource(s) in a functional manner, using reactive event driven APIs so, for example applying "Person" class to "Employee" role could shield a Resource set of people being working for someone. The ultimate goal is to be able to "plug" as much "backends" connectors as posible into distributed peers which exposes protocols / APIs for knowledge driven hypermedia applications.

Implementation. Async / Reactive Service URIs / Connectors (sample):

Spark,

Lucene / Solr,

Kafka,

MQ,

ServiceMix,

Vert.x,

OSGi,

Spring Boot,

SCDF,

Jena, (RDFS, OWL, Turtle, N3, SPARQL),

Reasoning / Shapes,

JAF / JCA / JDBC / JNDI / JMX,

Metamodel / Teiid / D2RQ / OData,

HAL,

DIDs,

OpenShift (containers / deployment).

Application:

features / techniques / patterns.

Implementation deployment use cases. Sample Apps: SoLiD / PIM / PASCEN: App declaratively built with framework, Implementation Integrations.

[1] <https://en.wikipedia.org/wiki/Data,_context_and_interaction>