* Meta Circular Interpreter (LISP Book):
* Encoding I/O: Graph Quads. Data. Ternary / Sets Representation / Aggregations Encodings in terms of Statements / reified Statements:
* (Context, Object, Attribute, Value);
* (Node, Type, Arc, Node);
* Layers / Services Resources Dataflow: Functional Statements Events / IO.
* Platform Schema: Model Layers, Components, Services, Protocols defined in same terms than Model Applications instances.
* Quads Graphs: Definition (Predicates) Roles / Assertion (Flows) Roles.
* Statements: Monads / Functors. Parse / Browse Statements: Definition / Assertion Roles.
* Platform / Application Services / Protocols:
* Model Platform.
* Model Application.
* Augmentation: Platform / Application I/O. Platform Augmented.
* According Input / Mapping / Output Ternary Roles (Reification):
* Ternary Definition: Statement : (Statement, Statement, Starement);
* Ternary Assertion: Statement : (Statement, Statement, Starement);
* Dataflow / Execution: Functional Mappings. Ternary Monads / Functors: Predicates Assertions / Transforms Flows / Browse Pipelines.
* I/O. Build:
* MessageSet(context : MessageSet payload : MessageSet);
* Context: current knowledge state.
* Payload: current request state.
* Eval input against context items:
* result : context.visit(input);
* Eval:
* Assertion result : CRUD / Branch.
* Flow result: request: context, Jmp.
* Ternary Mappings from Input Statements.
* Ternary layer to Schema.
* Schema layer to Behavior.
* Behavior layer backs forth until Augmentated Ternary Mappings.
* SAILs Layers Augmentation Protocols Services:
* Alignments: Data Matching.
* Aggreggations: Schema Matching.
* Activations: Behavior Matchings.
* Layer Matching Services Encodings:
* Data: Ternary Mappings.
* Schema: Sets. Kind Reification.
* Behavior: Quads Statements.
* Services: Dataflow Facade. SAILs Resolution / Inference Interface. Functional REST API.
* SAILs Layers Protocol:
* Encode REST CRUD / HATEOAS / DOM APIs:
* MessageSet(context : MessageSet payload : MessageSet) : visit : MessageSet;
* MessageSet::onVisit(MessageSet dialog) : MessageSet;
* Init result MessageSet:
* result : MessageSet(context, payload)
* Visit: each ctx item, visits each payload item recursively. Visits performs pairwise Augmentations. New statements / resources are added to the list of result pending evaluations context, removed if complete.
* Pendings: context. Visit may add resources if they are new with some arc(s) pointing to. Update (delete unbound arcs), delete (if no more arcs points to).
* Loop: reference URNs / forge new Resource / Statements API. Updated MessageSet contexts.
* Visit ends there no left nothing for further processing in context.
* forEach result ctx
* forEach  resource item
* Augmentations:
* [Augmentation.Alignment](http://augmentation.alignment)(ctx, item); Data Matching. Example: add rdfs:sameAs.
* Augmentation. Aggregation(ctx, item); Schema Matching. Example: adds Statements, Kinds, Sets / Reified Kinds Resources Aggregation. Reified rdfs:rdfType, rdfs:rdfClass, rdfs:subClassOf.
* [Augmentation.Activation](http://augmentation.activation)(ctx, item); Dimensional Measures / Discrete Relationship situations. Roles / Mappings: Events Modelling.
* Dimensional Example:
* (Dimension, Measure, Unit, Value);
* (Time, 1m, secs, 60secs);
* Discrete Example:
* (Relationship, Relation, Kind, Entity);
* (Marriage, aMarriage, Spouse, Jane);
* Add / remove payload. Next.
* Add / remove ctx. Next..

Básicamente necesito ofrecerme y estar disponible para diversos tipos de circunstancias que ahora se me ocurren:

1) Estar al día con lo relacionado a proyectos en marcha o ya desplegados y que pudiesen llegar a necesitar desarrollo eventual (plataformas, frameworks, arquitecturas)

2) Estar al día con lo relacionado a cómo se plantearía una solución ante una eventual propuesta y sus requerimientos (plataformas, frameworks, arquitecturas).

3) Plantear una serie de arquetipos y modelo de componentes lo suficientemente "genéricos" como para "generar" el esqueleto de las capas de una aplicación, su dominio y sus servicios: Apache Maven POMs, similar a Spring Boot pero con descriptores específicos para cada dominio y sus requerimientos.

Esto último lo estoy investigando y me parece factible: imaginate ERP, BPM, CMS, BI, etc. "descriptos" (Domain Driven Development like: StratML / Apache Isis examples) y Ontology Matching "Alignments" disparando "eventos" entre ellos y sus dominios (BPMs con ERPs y SCMs de diferentes dominios: incremento del precio del barril de petróleo, por ejemplo).

Cómo sea, sólo se me ocurre esto ahora por necesitar bastante el ponerme al día con todo. Estoy un poco oxidado. Ya se que es cuestión de arrancar y hoy más que nunca necesito tener la cabeza ocupada en algo útil.

Por esto último, me parece justo y para no generarme ansiedad, hacer las tareas (estas o las que se necesiten) en una primera fase de "inducción": básicamente actualizarme, estudiar código o aprender frameworks, sin pretender remuneración por ahora.

Entonces sí, cómo vos me pedís, creo que voy a estar en condiciones de saber cuándo te voy a poder decir que puedo arrancar.

Abrazo,

Seba.

Se me ocurre que sería posible retomar lo de las vidrieras "inteligentes". Un poco de ML, un poco de Semántica, un poco de Rules Stack (Drools) y Domain Driven Development Augmenting de, por ejemplo, un motor de recomendaciones / customer care / profiling / retention contextual aware (alignment de stakeholders en contextos: occurrencias de eventos o condiciones como el clima o del último campón del torneo de fútbol, ofertas de la temporada, modas, etc. StratML, ver adjunto).

Hay ontologías (SW Graphs "schemas" para esto). Y la abstracción: "consumidor, producto, bien, necesidad, productor, contexto" homologa una vasta serie de dominios e interacciones (DCI: Data, Context, Interactions) con features más que interesantes para inferencias y tensores para predicciones / clasificaciones (ANN deep classifications, regressions, clusterings). Ver adjunto.

Lo cortés no quita lo valiente. Hay varios grossos en Deep Learning, dicen, que demuestran como grafos (la abstracción básica de las tres capas de augmentation de conocimiento que propongo y sus componentes) son adecuados para Deep Learning trainings and predictions:

Augmentations (Data, Schema, Behaviors) matchings:

Alignment: Data Matching. Index Svc.

Activation: Schema Matching. Naming Svc.

Aggregation: Behavior Matching. Registry Svc. Continuos (Dimensional) and Discrete (Relationships) Models.

Plantear una serie de arquetipos y modelo de componentes lo suficientemente "genéricos" como para "generar" el esqueleto de las capas de una aplicación, su dominio y sus servicios: Apache Maven POMs, generados de manera similar a Spring Boot pero con descriptores específicos para cada dominio y sus requerimientos.

Que estos tres tipos de servicios y las capas de almacenamiento (SAILs) facilitan en Eclipse rdf4j una architecture de capas con IoC y separation de concerns como declarative extensions and protocols aware pluggability permiten Event Driven / Reactive Dataflows "smart routing" pipelines deployments.

Abstraer el denominador común (CDI Spring / DCI Data, Context, Interactions) para los servicios generales para cada tipo y dominio de negocio y aplicación (vía upper ontology / layers aggregated ontology) permita tener componentes que se "entienden" en base a domain driven knowledge shared (aligned) protocols.

De ser factible, vía Data, Schema / Behavior Alignments, imaginate ERP, BPM, CMS, BI, etc. "descriptos" (Domain Driven Development like: StratML / Apache Isis example) Repository Matching "Alignments" disparando "eventos" entre ellos y sus dominios.

BPMs con ERPs y SCMs de diferentes dominios, eventos y reactive behavior "semantically stated" : incremento del precio del barril de petróleo y los datos / contextos e interacciones que eso conlleve en cada caso: aumento de las reservas de combustible, por ejemplo. Ver archivo adjunto.

Básicamente necesito ofrecerme y estar disponible para diversos tipos de circunstancias que ahora se me ocurren:

1) Estar al día con lo relacionado a proyectos en marcha o ya desplegados y que pudiesen llegar a necesitar desarrollo eventual (plataformas, frameworks, arquitecturas)

2) Estar al día con lo relacionado a cómo se plantearía una solución ante una eventual propuesta y sus requerimientos (plataformas, frameworks, arquitecturas).

3) Plantear una serie de arquetipos y modelo de componentes lo suficientemente "genéricos" como para "generar" el esqueleto de las capas de una aplicación, su dominio y sus servicios: Apache Maven POMs, similar a Spring Boot pero con descriptores específicos para cada dominio y sus requerimientos.

Esto último lo estoy investigando y me parece factible: imaginate ERP, BPM, CMS, BI, etc. "descriptos" (Domain Driven Development like: StratML / Apache Isis examples) y Ontology Matching "Alignments" disparando "eventos" entre ellos y sus dominios (BPMs con ERPs y SCMs de diferentes dominios: incremento del precio del barril de petróleo, por ejemplo).

Links:

DGI: https://arxiv.org/abs/1809.10341

Archetypes: http://maven.apache.org/guides/introduction/introduction-to-archetypes.html

DCI: https://en.wikipedia.org/wiki/Data,\_context\_and\_interaction

CDI (IoC): https://www.baeldung.com/java-ee-cdi

Reactive Microservices: https://vertx.io/

SAILs (Layered Repositories): https://rdf4j.org/

https://en.wikipedia.org/wiki/Linked\_data

https://www.schema.org

https://solidproject.org/

https://lists.w3.org/Archives/Public/semantic-web/2020Aug/0037.html

https://github.com/sebxama/scrapbook

https://en.m.wikipedia.org/wiki/Strategy\_Markup\_Language (ver adjunto)

Saludos,

Sebastián.