**Contents**

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).  
  
Layered models semantic infrastructure for integration of heterogeneous backends (meta models).

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.  
  
Integrated view. Navigate contexts, data, interactions. APIs. Dimensional views annotations (analysis / mining).  
  
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.

**Features**

Graph encoding of data / schema / behavior. Dimensional / Grammar annotations. MetaGraph: augmentation / transforms (Messages). Features.

Parallel distributed graphs models augmentation / transforms synchronization (Messages). Event sourcing (distributed inferences). P2P / DIDs.

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

URIs API for annotating network retrievable 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.

**RDF triples, quads introduction**

RDF Models: rdfs type, class, subClassOf, sameAs, reification when appropriate. RDFS. OWL (alignments).

RDF / OWL Backend: APIs. Details: Contents triples / models introductions.

Turtle. N3.

Example: feed Dimensional model for equivalences (units), comparison (orders).

TBD.

**Models: Quads, Contexts, Occurrences, Attributes, Values.**

Declarative means of using RDF quads to state application object models (data, schema and behavior).

Aggregation.  
Kinds.  
Grammar.

Formalization: Functional / Object API. Reference / Data model. Sets, categories, models.

Subjects: attributes / values, contexts / roles.

(Context, Occurrence, Attribute, Value);  
(Context, Sign, Concept, Object);

Instance, occurrence, class, metaclass.

Hierarchies: layered quad statements are represented by a class hierarchy which root is the Resource<T> monad. There is a subclass relationship between each layer implementing class and the one of the next layer (Dynamic Object Model).

Quads in the context role of lower layers represents occurrences of context enclosing layer. Assert class hierarchies, order relation (temporal, causal, containment, etc.) by attrs / vals, set / superset relations.  
Discovery: All model kinds are browseable / discoverable.

Determine class (reified layers contexts) hierarchies:  
(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model resources, statements, kinds. Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch,  
event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.  
Dimensional / Grammar models.

TBD.

**URIs, Resource, Statement, Kind APIs**

TBD.

Message service URIs: contextual (statement / dialog) service invocations.

Example: Subject (image URI / resource : source), Predicate (detection service / index service), Object (detection / search results endpoint / placeholder : destination).

Grammars: Predicate Kind (face / search recognition signature) from Subject (faces images / names) / Object (face classes / subjects) Kinds. Kind model layers.

Models definition: data (Statement, Entity), schema / context (Role, Class), interactions / behavior (Flow, Behavior).

Kinds / Roles:  
Grammar: kinds layers aggregation (CSPO layers Kinds).  
Layers: Roles (Models metaclass context resources).

Reified Kind: (Kind, Occurrence, Attribute, Value);  
Grammar input set model specificatíon (Statement layer kinds).

Dimensional input set model specificatíon (from Statement layer, ordered SPOs: order criteria, comparisons. Kinds / class / occurrence / instance  
order criteria?). Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements).

MetaGraph (resolution). Dimensional / Grammar alignments / annotations.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch,  
event bus routes. URIs / IDs mappings. Resource set specification resolution. MetaGraph resolves concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics via MetaGraph driven transforms (data / schema / behavior augmentation: dialogs).

URIs API for annotating network retrievable 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.

Resource<T : URI> monad. Message functors. Transform reactive extensions.

Transform : Observer / Observable of Resource<T : URI>. Stream. Built upon Resources / Messages (TransformBuilder).

Identity and other core transforms (core messages). Stream.

flatMap(Message::apply) : Transform<Resource<R : URI>>.

API: Class for layer for model.  
API: Class for layer (DOM).  
API: Parameterized Resource: layer classes determined by URIs hierarchy, i.e.: Resource<Entity>, Entity : URI.

Base core services URIs (index, naming, registry). URI subclasses implementing / wrapping state for Resource monads offering protocols / addressing / content types / representations facades for services: DBs, WS (REST, SOAP, SPARQL), ML (predictions), etc.

Discovery: All model kinds are browseable / discoverable.

Determine class (reified layers contexts) hierarchies:  
(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model resources, statements, kinds.

Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

MetaGraph (resolution). Dimensional alignments / annotations.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

**Model Layers**

See Messages / Augmentation.

Composed of quads semantically aggregated into layers.

Core features provides:  
Alignment  
Activation  
Aggregation

Message / Transform driven specification of Alignment, Activation, Aggregation (Augmentation).

Message (Resource set expression);

Message: Model parent layer (Resource). Nested Messages CSPOs.

Model: Message scopes. Described as (nested) Message Resource set expressions.

Models: Data (Models), Contexts (Grammars), Interactions (MetaGraph : Models / Grammars bindings). Dimensional annotations.

MetaGraph: Resource, Statement, Kind class / instance as CSPO MetaGraph statement roles. Class / subclass relationship, Kind / subkind relationship.

Reactive nodes (Message events):

. Input Message event;

. Augmentation;

. Model / MetaGraph Message resolution (grammars / models / backends / services); Model Resource(s) response activation;

. Augmentation (Message : response / dialog);

. Output Message (events);

Grammars.

Upper / Dimensional ontology.

Inter models alignments.

Services (Endpoint URIs: Resource facades).

Reified model resources (CSPO, Resource, Statement, Kinds, Layers). Augmentation (Alignment, Activation, Aggregation) Messages / Transforms.

Model, Contexts, Interactions IO:  
Model: aggregated resource statements.

Context: aggregated model kinds (grammar statements).

Interaction: aggregated model / context bindings.

Inputs: resource statements, resolvable messages. Operation semantics (CRUD, browse, etc.) according input statements layout. Model endpoint. Materializes input resource statements and fully resolved message resource statements from interactions applying Augmentation and matching messages transforms.

Resource flow: input plain RDF URIs statements. Model / Context updates. Transform matches concrete resources.  
Resource flow: input message URIs statements. Context / Interaction perform. Transform matches resources in messages context grammar kinds hierarchies.

Outputs: resource statements with possible further resolvable messages (Model IO recursion / dialogs). Interaction queries context / model back for further resolutions. Message transform stream with request message applied plus matching context resolved resources from message.

Model, Contexts, Interactions IO:

Resource, Statement, Kind, Message, Transform.

Subscription, Subscriber, Producer, Consumer, Processor.

Model: aggregated resource statements model.

Context: aggregated model kinds (grammar statements model).

Interaction: aggregated model / context / dialogs bindings model.

Aggregation: layers. Parameterized Resource<C, S, P, O> : CSPO : URIs hierarchy.

Materialized interactions re-populate model and context (Augmentation). Browse context model: kinds and grammar known statement "templates" (by kinds hierarchy layers aggregation) navigation for discovery of domain messages resource kinds.

Model, Context, Interaction IO: Message. Nested CSPO contexts quad, CSPO resources (plain URIs, kinds, nested contexts). Wildcards, variables, placeholder, null values: Message structure defines CRUD behavior.

Message: Resource model hierarchy parent class (monad of plain URI, parameterized resources). Resource set specification. Any Resource is a Message, specifying a potential set of other Message (Resource) in a model (layers).

Resource : Message. Resource resolution: known URIs, known resource kinds bindings, dialog (resource set specification) recursively. Interaction model (dialog resolved resources set). Wildcards, variables, placeholder, null values: Message structure defines CRUD behavior.

Resource monad of URIs or Message monad of Resource?

**Data Model**

Data Model layers population / augmentation.

(Resource, Resource, Resource, Resource) : Resource / Message (Model).  
(Entity, Subject, Attribute, Value);  
(Role, Entity, Attribute, Value);  
(Kind, Role, Entity, Attribute);  
(Class, Kind, Role, Entity);  
(Flow, Class, Role, Entity);  
(Behavior, Flow, Class, Role);

Messages (Model : Resource) as Resource set specifications. Subject, Attribute, Value : Resource.

Determine class (reified layers contexts) hierarchies:  
(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model, resources, statements, kinds.  
Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch,  
event bus routes. URIs / IDs mappings. Resource set specification resolution. MetaGraph resolves concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics via MetaGraph driven transforms (data / schema / behavior augmentation: dialogs).

Models definition: data (Statement, Entity), schema / contex (Role, Class), interactions / behavior (Flow, Behavior).

Kinds / Roles:  
Grammar: kinds layers aggregation (CSPO layers Kinds).  
Layers: Roles (Models metaclass context resources).  
Reified Kind: (Kind, Occurrence, Attribute, Value);

**Schema Model (Grammars)**

Schema Model layers population / augmentation.

Grammar Resource input set model specificatíon (Statement layer kinds Messages).

Grammars: Predicate Kind from Subject / Object Kind. Kind model layers.

**Behavior Model (Dimensional annotations)**

Dimensional Model layers population / augmentation. Purpose modelling. Dimensional Concepts.

Order layers statements. Hierarchies (contexts / kinds). Parent / child relationships (steps). Order type relationships: husband: single / marriage / married.

(Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.  
(Measure, Value, Previous, Distance);  
(Unit, Measure, Value, Previous);  
(Dimension, Unit, Measure, Value);  
(Concept, Dimension, Unit, Measure);  
(Resource, Concept, Dimension, Unit);  
(Statement, Resource, Concept, Dimension);

Populate / align / annotate models with dimensional data. Model input: statements (model resources). Model specification: augment, sort  
statements. Model specification: specialization of base model layers. Resolve resolution statements order.

Dimensional input set model specificatíon (from Statement layer, ordered SPOs: order criteria, comparisons. Kinds / class / occurrence / instance  
order criteria?).

Value, Previous, Distance, Next. Dimension, Unit, Measure, Value (aggregated ordered statements layers).

Value -> distance(prev, next); ordering;

Assert knowledge: 1h -> 60min;

dom-lun-mar-mie-jue-vie-sab (orders);

1mt -> 100cm;

etc.

Comparison / order: Alignments (prev, curr, next asserted knowledge). Next hour, location, city, country, next distance at next time at current speed. Event sourcing / tracking: married -> marriage occurred.

Sort: cause / effect, temporal, etc. Messages align, functional map, fold, etc. Primitives. Encode layered statements ordering. Complement / supplement concepts definitions.

Events metamodel (TBD):

(Object, State, Axis, Type)  
(State, Axis, Type, Event)  
(Axis, Type, Event, Event)  
(Type, Event, Event, Event)  
(Event, Event, Event, Event)

**MetaGraph Model (models aggregations)**

See Message Resolution.

Model MetaGraph (TBD):

MetaGraph: Resource, Statement, Kind class / instance as CSPO MetaGraph statement roles. Class / subclass relationship, Kind / subkind relationship. Grammar / Model bindings.

Grammar: layers aggregate kinds from resource / statement layer or kinds for each model layers.

Layers, contexts, occurrences, kinds: Role Entity layer occurrences instantiated with each Entity SPO as Entity subject (Entities occurrences in Role context for each Entity SPO). Idem for subsequent layers.

Statement class: context.

Statement instance: context occurrence.

State resource kind in occurrence in context.

State resource (context) class / (occurrence) kind hierarchies.

State Resource URIs occurrences / class IDs.

Resolve Message matching Resource from behavior layers / matching kinds from Model / data layers.

(Kind, SuperKind, Attribute, Value);

(Occurrence, Kind, SuperKind, Attribute);

(Context, Occurrence, Kind, SuperKind); (attributes / links bindings).

(Resource, Context, Occurrence, Kind); State Resource Kind in occurrence context (context / role bindings).

(Statement, Resource, Context, Occurrence); State Resource URIs occurrences / Resource class IDs (classification bindings).

(Interaction, Statement, Resource, Context);

(Action, Interaction, Statement, Resource);

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values.

Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

Ontology matching (table, pk, col, val example). Helper upper models for models linking / alignment.

Events declarative definition. State change of value in axis in measure of context.

Messaging metamodel:

(Message, Resource, LHS, RHS);  
(Interaction, Message, Resource, LHS);  
(Role, Interaction, Message, Resource);  
(Context, Role, Interaction, Message);  
(Dataflow, Context, Role, Interaction);

**Datasources / Backends / Services (URIs)**

TBD.

**Addressing. IDs. Encodings**

Resource<T : URI> monadic hierarchy.

Basic hypermedia browse / CRUD (HTTP verbs) bound Message functors compatible for all Resources (REST).

Resource.flatMap(Message::apply) : Observable<Resource> (stream). Composable functions.

Basic Message application (Context Mapping): shift right mapped applied statement resources. Mapped resource context > instance (occurrence) of next layer message reified resource context.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings. Resource set specification resolution. MetaGraph resolves concrete resources, Message expansion.

Resolve Message / dialog (CRUD) semantics via MetaGraph driven transforms (data / schema / behavior augmentation: dialogs).

Dimensional / Grammar models.

aX^4 + bY^3 + cZ^2 = dW;  
d, a, b, c: classes (CSPO);  
WXYZ: instances (CSPO);  
Powers: CSPO role;  
Terms: CSPO resources;  
Z(obj) is Y(pred) for X(subj) in W(ctx);

Instance, class, metaclass, occurrence terms. Primitives, variables, placeholders.

Resolution (Discovery, DIDs). Templates (grammars). Subjects: attr / val, ctx / role.

Behavior: order / compare.

Proof of work.

MetaGraph model: map URIs -> IDs.

Satisfy dW. Sync resolution (recurse terms contexts).

FCA. Resource attributes.

Tensor, adjacency matrix, tree.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings.Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics. Dimensional / Grammar models.

Naming: Context URIs. Dimensional (Statement, Resource, Kind) addressing (conventions). Discovery: patterns / locators: Semantic URIs / MetaGraph ID mappings. Encodings: contextually encoded addresses / URIs.

Naming: NLP. Bind / suggest human readable names / labels.

Naming: Source (plain class) URIs.

Naming: Statement (Context) addresses.

Naming: Occurrence URIs (in Statement in CSPO role).

Naming: Contextually encoded addresses (URIs in Occurrences in Statements in relation with other occurrences).

Naming: Kinds addresses (global / mask, from occurrences in statements). Signatures. MetaGraph: operate over IDs.

TBD.

**Dataflow (reactive models)**

TBD.

For input Resource(s) (Model reactive / async IO APIs):  
. Create / retrieve Model  
. Create / retrieve Context Message(s)  
. Create / retrieve Message(s) Interactions  
. Bind Interaction Message Resource(s)  
. Perform Message transform. Materialize results. Message application rules: upper / domain ontology selectors (closest matching role in  
hierarchies), context alignments.

Match request statement / graph with model via context in interaction (algorithm: addressing, encoding, interaction model upper bindings /  
alignments). Resource MetaGraph. Reified model resources (Resource, Statement, Kinds, CSPO, etc.).

Apply subsequent transforms in interaction context (referrer context, get classes playing entity role, get behavior flows, browse / navigate  
streams). Context, variables, wildcards, placeholders.

Services: distributed addressing / resolution, reactive distributed event bus: streams / contracts, index, naming, registry.

Discovery: All model kinds are browseable / discoverable.

Determine class (reified layers contexts) hierarchies:

(ContextReifiedClass, ContextReifiedSubClass, SubClassAttributeKind, SubClassValueKind);

Merge / specify model, context, interaction graphs. Reified model resources, statements, kinds.

Model, context, interaction model graphs layers specifications. Reified models layers contexts resources describe graphs. Augmentation. Message context statement occurrence: Model.

Message flow (event loop) in / out:

Activation (data) <-> Alignment (schema) <-> Aggregation (behavior);

Encode behavior in statements / graph:  
Comparisons, order. Sort. Order (kinds hierarchy?).

Pattern matching, iteration, jumps. Discovery: routes / signatures, next event in bus / graph.

Context Model Message: Resource Specification (Grammar Template).

Messages Model: context model instance from input model grammar. Transform: context model instance from Messages.

Express Augmentation (Activation, Alignment, Aggregation) as Messages / Transforms. Reified Model entity types / roles (CSPO, Kinds, Layers, etc.).

Resource monad of URIs or Message monad of Resource?

Encoding. Addressing. Schema / MetaModel for data (Model), schema (Context), behavior (Interaction) resources / layers (aggregation). Naming  
formats / schemes: namespaces, contexts.

Class hierarchies (express context / class / kinds hierarchy). Grammars / Dimensional metadata.

Resource MetaGraph bindings (Message expansion / resolution index).

Subscription, Subscriber, Producer, Consumer, Processor. Example: submitting Behavior layer grammar / context "template" initiates "dialog" for fulfill Behavior expanding Message(s) and nested context layer statements (known / resolvable, new behavior / subitems) needed to complete / update full Behavior layers contexts graph.  
Augment. Alignment, Activation, Aggregation Message(s) : Resource set specifications.

Model listens onMessage (interaction context model population / dialogs scopes / namespaces).

Model augments input Message (augmentation specifications over in Message).

Model expands Message (Message over model resources):

Resource listen modelMessage. Model subscribes to response.

Matching triggered Resource. Message matching semantics (transforms).

Triggered Resource publish itself modelMessage.

Model augments output Message (augmentation specifications over out Message).

Model publish onMessage (interaction context model dialogs / resource dumps).

**Messages: Transforms. Graph Execution Semantics**

Message encoding semantics resolve transform execution resource set declaratively from MetaGraph / Models.

Specification resolves to query / create / update / delete according interaction contexts. Messages models determines “possible” messages according models grammars. Interaction specifications (statement / graph / dialog) may have any message encoding components in corresponding statement roles.

For each behavior, flow, class, kind, entity, statement in input request, transforms matches those components by applying messages into model resources (grammar) matched into interaction model (binding subsequent roles by dialogs).

New (potentially unknown) resources are added and augmented into the graph. Augmented resource events emitted from transform streams.

Example: a message composed of a kinds CSPO matches statements “instances” of those specifications (statements whose CSPO have matching kinds). A message with three CSP kinds and a (potentially unknown) object URI retrieves matching resources having that object value into corresponding property kinds. An statement of plain (potentially unknown) URIs instantiates / updates and augments new / known resources added to models and returns an augmentation transform result.

Interaction Model: Context of Messages model for a given interactions session / dialog state. Message invocation requests: Statement(s) building Resource invocation graph with layers matching Message patterns. Layers graph invocation patterns matching from higher to lower layers resources fulfilling higher layers templates. Variables, wildcards, placeholders.

Dialog arguments resolutions example: higher layer Resource / Message request / invocation instantiates in Interaction Transform context corresponding lower layer graph statements to be “populated” to fulfill request. Message IO of “forms” (Messages) inter-peers (originating peer  
acting as “server”) for initial requested peer to “ask” for form elements to be populated (interaction context “dialogs”). Resolution may propagate to other peers (content aware addressing dataflow routes dispatch: P2P resources address encodings, matching forms models requests). Nested interactions.

Explain messages (resource resolution). Grammar. Match model Resource(s). Compound nested CSPO statement contexts defines result behaviors. Message CSPO contexts may define create, retrieve, update or delete operations (passing 'null' for example for resource / statement to be deleted).

Explain transforms (message application). Transform: Resource stream result of Message application over resolved Resource(s)). Input statements: Message(s) / Resource(s) (from input message or to be populated or populated in dialog) and "goal" Message / Resource aggregating a model from Resource MetaGraph with Message / Resource bindings.

Message types (Augmentation: onto / domains):  
Attribute / Link (data):  
. Alignment: Augment / infer Attribute / Link.  
Class / ID (schema):  
. Activation: Augment / infer Kind, Class.  
Role / Context (behavior):  
. Aggregation: Augment / infer Role / Context.

Runtime / Resources / Messages: Core (upper / onto) Resources, Messages, Transforms. Reified entities (CSPO, Kind, SubjectKind, etc.). Match cases in messages.

Core (upper / onto) Messages: Getters, setters, nav, etc.

Domain Messages: raiseSal: setSal(sal \* increment); promotion: setPosition.

Event sourcing / tracking: married -> marriage occurred.

Resource.flatMap(messageInst::apply) : Resource.

Dataflow: Messages hierarchy. Aggregate contexts from coarse to fine grained transforms (raiseSal -> setAttr).  
data <-> schema <-> behavior.

Message dispatch, input statements resolve to applicable messages from switch from behavior to data layer invoking async microservice.  
Message case matching may involve entering and leaving data, schema and behavior paths if aggregated contexts matches more than one  
message. Visitor.

Message: functor (monadic transform) : Resource<T> -> R, T, R : URIs (hierarchies, models, semantic content types). Available verbs / flows /  
navigation (browse models, state of application returned from materialized models). Parameterized functions (partial applications) into Messages metamodel resources. Contexts (dataflow). Execution graph.

Alignment Message: Resource -> Statements (attributes, values).  
Activation Message: Statement -> Kind, Class.  
Aggregation Message: Statement -> Statement (next layer).

Subscriptions declarations / definitions. Applied on streams activations (transforms, executions resource parameterized partial contexts).

Messages metamodel: functor declarations partially defined over metamodels resource (T) defining transforms into (R) over application  
(flatMap) over / into (S). Messages inferred / aligned, activated, aggregated according base message transforms resources. Messages inferred from models / layers. TBD.

Functors <T, R> -> Resource<R>

Form / Template describing (reified as a Resource in a context model) declaratively subscriptions and actual exchange capabilities (datflow).  
Mappings, Transforms.

Processor which acts upon Resource events. Materialize results.

Specify declaratively augmentations by means of messages.

Upper onto / domain aggregated messages.

Event bus: P2P deployment.

Messages: Monadic applicables over Resource (flatMap).

Base HTTP / Browse (REST) Messages. Custom Messages.

Model MetaGraph: Resource, Statement, Kind (reifying class / instances) contexts / occurrences / attributes / values. Encoding. Message dispatch, event bus routes. URIs / IDs mappings. Resource set specification resolution. Resolve concrete resources, Message expansion. Resolve Message / dialog (CRUD) semantics.

**Augmentation (via Messages)**

Activation (Statement / Entities : data).

Alignment (Kinds / Classes : context / schema).

Aggregation (Flows / Behaviors : interaction).

Messages describes declaratively augmentation steps materializing models contexts / hierarchy layers.

**Protocol (API): dialogs (distributed resource augmentation / sync)**

Message resolution (contexts).

Reactive. Interaction / session contexts.

Annotate, link, browse resources instances, classes, metaclasses, occurrences in roles in contexts, attributes / values. Services / clients: endpoints: Virtualization (wrapper protocols).

Semantically annotated content types: image/png;face, text/xml;faceImgCoords. RDF schemas describing content, attributes, links in context / target roles. Content types: labels (schemas).

Message: Context Model API. Input statements: Model Grammar. Augmented IO by interaction transforms of applied matching Message with model statements inputs. Context of core models instances. API.

Transform: Interaction Model API. Input statements: Transform request invocation specification. Functional application of Message(s) over Resource(s): Transform (streams). Augmented IO: Requested Transform which applied augments resulting responses (dialog arguments  
resolutions). Context of context model instances.

Reactive / streams API. Message Transform (interaction result): matches request context specification built upon Resources / Messages (TransformBuilder). Resolve state / dialog session graph. Returns observable stream. Dataflow (chaining). Operations (over streams).

Transform request invocation specifications: means to interact with underlying contexts models (CRUD, domains behavior). Transforms result from applicating Message(s) over Resource(s). Sending a Message Resource to a given interaction context initiates a “dialog” in which to “populate” target Resource(s) and Resource arguments. Each dialog “step” renders resources / layers streams of requested arguments (server “queries” clients) or resources / layers streams of response augmented Resource(s).

Graph linking / alignment / synchronization by entailments from event sourcing over inferred state. Distributed predictive alignments.

DOM / OGM APIs (JAF). I/O Implementation, Deployment.  
Model, URI, Resource, Statement, Kind hierarchies. Models architecture (URI class per layer). DIDs / P2P / Rx Implementations. Model API. ModelManager. Event loop. IO.

**Protocol (API): resource activation (hypermedia application browser)**

Reactive. Interaction / session contexts.

Protocols / Services / Clients: Context interaction sessions (state flows).

Content type activation. Messages / gestures. Rules (commands / verbs). Content types: labels (schemas).

Browser referring context (Work, Peter, Employee).

Annotations (protocol): JSON-LD. Model / Grammar / Dimensional. Map annotations to resources (query string / meta resource description). Browse data (model), schema (grammar), behavior (metagraph).

Models ‘plug’ into Runtime augmenting its capabilities via standard extension APIs (added features / knowledge reactive URIs). Models ‘modules’: parsing modules declarative descriptions. Augment, link instance data.

Upper aligned ontology plugins / blueprints:

Resource URIs specialized implementations for different connectors / endpoints and content types (DB / OData, REST / HAL, etc.). Feature  
Resources backends (i.e.: URI for DB interaction).  
Purposes: Metamodel declarative goal statement. Fulfill flows (templates / forms: Messages).

Goal: P2P service that connects to services / endpoints (DB, REST, etc.), homogenizes them and exposes an API by which (augmented)  
knowledge of an stated entity is returned in response (protocol that entails queries / CRUD, object navigation in message / session state contexts). Peer shares / syncs with other peers.

Goal: Intermediate API (HAL for example) aggregating previous objects knowledge (DCI, DOM, OGM, MVC)

Goal: Semantic Browser. Homogenize diverse domains. Query examples. Search session history. Referrer semantics. Collected items in goals roles. Create session purpose document. Link to / from any addressable resource in context / role. Annotate source / destination context roles,  
attributes and schema.

TBD.

**Ontology matching**

TBD.

**Data / Reference Model (APIs, Functional Semantics)**

TBD.

Upper ontology: Node "levels" of domains abstraction. Highest level: service / user interaction (resource / hypermedia activation: model gestures). Lowest levels: upper ontology / business domains.

Application / Site / Service node types (Node ontologies domains layers). Renderers producers / consumers. Backends integration (Augmentation, Messages).

**Platform: implementation**

Introduction. Document. Use Cases (EHR). Standards. Models (predictions / signatures).

Implementation. Languages. Backends. Reactive frameworks / microservices. Distributed consistency. P2P / DIDs. Models / APIs. Nodes / Endpoints. Containers. Deployment.

TBD.