**Distributed Consistency for Semantic Integration of Applications Knowledge**

**0: Outline**

Trust. Consistency. Event sourcing. Inferencing (of distributed state). Reconciliation. Certify Entity / Subject Identity. Class / instance alignment (matching).

Distributed systems / micro services access to shared data. Shared distributed data consistency / inference. Ontology matching. Integration (EAI / ESB). Introduction of new features / products integrating over existing (linked) data with Semantic capabilities and semantic enhancements.

Distributed Knowledge Base. Functional Syndicated Application Integration Framework. Plug existing backends (applications / datasources / services) via Connector(s) in an EAI / ESB fashion. Provide semantic augmentation of learned applications metadata (data / schema / behavior).

**0.1: Mision**

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 for applications interoperation

**0.2: Vision**

Domain Business Modelling. Integration. Syndication. General purpose business domains upper ontologies for ad-hoc application building overs existing domains. Domains: Use cases “problem spaces" domain translation / exchange / integration. Hypermedia Activation and Addressing. Link extended content types resources elements / parts with other resources addressed elements in a dimensional fashion.

**1.: Use Cases**

Integration by Augmentation:

Model I/O materialized in source (plugged) application / services backends. Framework inferences augment original (source) applications and serviced.

Integration by Extension:

Extended functionalities data / schema / behavior exposed as services external to source (plugged) applications. Sync (Augment / link) sources. Declaratively stated via Model descriptions. Discoverable, browseable (HAL / REST).

Declarative Application Design.

Semantic components.

BI / EAI smart dashboards / reports / workflows / process / activity / indicators inference / prediction / execution. Abstract upper ontology application models. QA, polls, learning, profiles, guided task wizards / editors. Goal. Purpose. Forms. Templates. Model context to fulfill (roles / rels).

**1.1: Problem**

Distributed systems / micro services access to shared data. Shared data consistency. Ontology matching. Integration (EAI / ESB). Introduction of new features / products integrating over existing (linked) data with Semantic capabilities and enhancements. Domains: use cases “problem "spaces" (distributed domain translation / exchanges / integration).

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.

**1.2.: Solution**

Hypermedia Use Cases (Ontology Levels). Integration by Augmentation: Aggregation / Alignment / Activation Annotation of distributed resources. Augmented Content types driven. Encoding / Addressing (links / browse / parts / rels / roles). Microformats (embedding). Wiki like abstract interfaces / representation (indexes).

Ontology matching (Data, Schema, Behavior alignments):

Data alignment:  
  
Determine if two instances (example: database records) of two different backends or services refer to the same entity / database row (Customers : John D. / Employees : John Doe).  
  
Schema alignment:  
  
Determine, for example, meaning and equivalences between diverse (aggregated / composite) schemas (equivalent classes / tables, equivalent attributes / columns, equivalent roles / relations).  
  
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).

Augmentation:

Augmentations: aggregate / align / activate (classify) sources of ontology matched data / schema / behavior enabling semantic layers interoperation.

Aggregation: Infer input data streams data, schema, behavior class / instance context layers.

Alignment: Infer layer missing / deducible attributes and values.

Activation: Infer layer CSPO Kind / Roles. Basic type system.

The idea is to infer Schema (classes and instance of classes / relations) operating over Data layer. Then, by aggregating Data and Schema, infer Behavior (classes and instances of operations / functions). This Aggregation, together with Alignment and type Activation mechanisms comprehend the Model core Augmentations.

Reactive Event driven Model Representations.

Purpose driven hypermedia activation:

Protocols / Services / Clients: Context interaction sessions (state flows). Content type activation. Messages / gestures. Rules (commands / verbs). Browser referring context (Work, Peter, Employee).

**2.: Approach**

Distributed P2P (Blockchain based) approach of semantic data persistence and synchronization between peers for ease of deployment patterns election and data sources integration (client APIs, microservices, etc.). Layered models semantic infrastructure for integration of heterogeneous backends (meta models).

Source inputs of Connector(s) (plugged backends, applications, datasources) and data coming from declaratively stated Model events interactions (Message IO) is rendered in a layered Model of RDF Quad Statement(s), each one aggregating input data into a Meta Model for further functional facet abstractions into data / schema / behavior layers, dimensional layers and semiotic layers.

Integration by augmentation: sources / back ends. Model I/O materialized in source (plugged) application / services backends. Framework inferences augment original (source) applications and services.

Integration by Extension: Extended functionalities data / schema / behavior exposed as services external to source (plugged) applications. Sync (Augment / link) sources. Declaratively stated via Model descriptions. Discoverable, browseable (HAL / REST).

Declarative Application Design.

Trust. Consistency. Event sourcing. Inferencing (of distributed state). Reconciliation.

Certify Entity / Subject Identity. Class / instance alignment (matching).

Integration: Augment sources / back ends. Model I/O materialized in source (plugged) application / services back ends.

Integration: Extension. Extended functionalities data / schema / behavior exposed as services external to source (plugged) applications. Sync (Augment). Declaratively stated via Model descriptions. Discoverable, browseable (HAL / REST).

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 (Hypermedia Application Language), OData (REST) like interfaces.

Hypermedia Activation. Addressing. Link extended content types resources elements / parts with other resources addressed elements. Domains: data, schema and behavior of business applications (ERP, CRM, BI, SCM, HMS, etc.) addressable information resources inference and resolution. 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.

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.

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.

**2.1.: Dataflow Augmentation KB**

Functional declarative way of stating Augmentation Transforms over Messages / Resources matching / populated by input Templates performing output Mappings Augmentation reflecting input, model and behavior state.

**2.2.: Reactive Event Driven Architecture**

Message based Augmentation Events Dataflow. Augmentation Mapping Dataflow allowing to embed dynamic state in Model entities (including Mappings Augmentations themselves).

**2.3.: Ontology Matching**

Determine whether two identifiers refer to the same entity or type, whether two relations are the same and which results corresponds to instances of the same actions.

Ontology matching. Dataflow: sort statements. Units. Equivalences. Distances / events (order). Services (Augmentation / Context Functors Meta Model mappings / transforms).

Explain ontology matching: data, schema, behavior alignments. Layers. Levels. Facets. Meta Resources / Model. IDs, Encoding / Addressing.

Encoding. Functional, Semiotic, Dimensional (Facets). Layers. Levels. Meta Resource / Model. Sets. Value as occurrence of attribute. metaclass / class / instance IDs.

Functional / Semiotic / Dimensional layers / levels examples / alignments.

Ontology matching (Data, Schema, Behavior alignments):

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 / tables, equivalent attributes / columns, equivalent roles / relations).  
  
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).

Model, URIs, Resource, Contexts Functional APIs. Meta Model / Resources encoding. Mappings.

IDs: Addressing / Encoding. Semantic (signature, contents, context) resolvable / discoverable identifiers.

Ontology Matching:

Ontology Matching: IDs, Addressing, Encoding. Functional, Semiotic, Dimensional (Facets). Layers. Levels. Meta Resource / Model. Sets. Value as occurrence of attribute. metaclass / class / instance IDs.

Ontology Matching: Encode: order, iteration, flows, units, relations, events, enums, etc.

**3.: RDF: Introduction**

**4.: RDF Quads / Object Mapping (DOM / OGM)**

As RDF Quads encodes four URI values (CSPO Statement) an Object - RDF Quad elemental mapping could be implemented regarding an RDF Quad Statement CSPO as follows:

(C: Context, S: Occurrence, P: Attribute, O: Value);

where Context (C) is the URI of an Object Class identifier, Occurrence (S) is the URI of an Object Class Instance identifier and, aggregating same Class / Instance pairs, Attribute (P) and Value (O) are, respectively, Class Instance member (name, domain / range) and values for the aggregated (S) Object of Class (C).

Contexts. Occurrences, Attributes, Values: Roles of Meta Resource(s) in contexts.

Subject in Statement has Predicate and Object Attribute / Value (roles).

Predicate in Statement has Subject and Object Attribute / Value (roles).

Object in Statement has Subject and Predicate Attribute / Value (roles).

Value as Occurrence of Attribute in Attribute Occurrence Context.

Context Kind (signature): Subject Kind and Object Kind Attribute / Value (roles).

Subject / Occurrence / Context / Role : Attribute, Value. Concepts. Semiotic Metamodel. Dimensional Encoding: each type as each (pair) kind. Pairs (tags / facets).

Meta Model: Layers Resource relations:

Instance, class, metaclass, occurrence, role. DOM, Actor / Context / Role.

Layer Context: Statement class. Aggregates same Context Statement(s). Next layer metaclass (Occurrence)..

Layer Occurrence: Statement Context metaclass. Aggregates same Context / Occurrence Statement(s). Previous layer context.

Layer Attribute: Statement Context Ocurrence Attribute (occurrence). Previous layer Occurrence.

Layer Value: Statement Context Occurrence Attribute Value (role). Previous layer Attribute.

Layer Aggregation begins with Model initial Statement having a new Context (class) “pushing” previous CSPO right, being the new class the new layer Context and CSP becoming SPO:

(C, S, P, O) : (N, C, S, P).

Functional / Object Oriented Resource API (Model, Statement, Semiotic, Dimensional layers, Meta Resources).

**5.: Models**

**5.1.: Contexts Quads Layers**

Kinds: example aggregate CSPO Kinds from an Statement. Kind / Class hierarchies. Order (dataflow / dimension / grammar / facets).

Layers: Data, Schema, Behavior. Class / Instance.

Interaction Layer.

The idea is to infer Schema (classes and instance of classes / relations) operating over Data layer. Then, by aggregating Data and Schema, infer Behavior (classes and instances of operations / functions). This Aggregation, together with Alignment and type Activation mechanisms comprehend the Model core Augmentations.

What my attempts where about in the beginning was to match different URIs or, for example, database 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.

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.

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 according the hierarchy layer level it corresponds (and declaratively stated in a Model of Meta Resources).  
  
Layers shape is as follow:  
Resource : Functional URI wrapper.

(Context : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);

CSPO Names are according roles (Meta Resource) in the Model. For example: layer Occurrence is parent layer class.  
  
Each layer abstract instances of its own contexts instances.

Input Layer: (CSPO layer):

(Transaction, someOne, buys, someProduct);

Statement (data layer instance):

Inputs regarding the same context are aggregated into data layer instance.

(Statement, Occurrence, Attribute, Value);  
(transactionStatement, someOne, buys someProduct);

Entity (data layer class):

Aggregated Statement and Occurrence Statement occurrences reified into an Entity along with its Occurrences Attributes.

(Entity, Statement, Occurrence, Attribute);

(someTransaction, transactionStatement, someOne, buys);

Role / Kind (schema layer instance):

Aggregated Entity and Statement Entity occurrences reified into a Role / Kind along with its Statements and Occurrences.

(Role / Kind, Entity, Statement, Occurrence);  
(someBuyer, someTransaction, transactionStatement, someOne);  
  
Class (schema layer class):

Aggregated Role and Entity Role occurrences reified into a Class along with its Entities and Statements.

(Class, Role, Entity, Statement);  
(Person, someBuyer, someTransaction, transactionStatement);  
  
Flow (behavior layer instance):

Aggregated Class and Role Class occurrences reified into a Flow along with its Roles and Entities.

(Flow, Class, Role, Entity);  
(someBuy, Person, someBuyer, someTransaction);  
  
Behavior (behavior layer class):

Aggregated Class and Role Class occurrences reified into a Behavior along with its Classes and Roles.

(Behavior, Flow, Class, Role);  
(Buy, someBuy, Person, someBuyer);

Then, each Model aggregates its Statements in the form (for example):

(Model Impl, Buy, someBuy, Person); Interaction / Meta Model.  
  
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 Meta Model. The act of applying an Augmentation implies one source Message Resource (context layer), one matching Template Resource (input signature) an Augmentation (Interaction functor) a Transform Resource (output signature) and a resulting (set of) Message Resource(s) materialized as further layers instances / Messages to be “parsed” by further corresponding Augmentations of matching Template signatures (dataflow).

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

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. Reactive Streams.

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.

Kinds (Application):

Kind: Basic type inference. Applied over layers CSPO during Activation Augmentation. An Occurrence Attributes / Values, aggregated for its URI and Context, determines Kind "members" (Attribute) and Kind instance member values (Value).

Super Kind / sub Kind hierarchy relationship is given by a set of Kind Attributes being super set / sub set of each other.

Examples.

SubjectKind (meta Resource): For a given URI occurring as Subject (Occurrence) across a set of Statements (Contexts), its aggregated Predicates (Attributes) defines its "Kind" and its Attribute values determines the given Kind instance "members" values.

ObjectKind (meta Resource): for a given URI occurring as Object (Value) over a set of Statements, Subject (Kind Attribute), Predicate (Kind Value).

PredicateKind (meta Resource): for a given URI occurring as Predicate over a set of Statements, Object (Kind Attribute), Subject (Kind Object).

ContextKind: SubjectKind (Attribute), ObjectKind (Value). Context (Statement) "signature" (dataflow inputs / outputs activation: domain / range).

OntResource is the class responsible for aggregating different URIs referring the same entities (Ontology Matching).

Resource : Functional (Monad) OntResource wrapper.

(Context : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);

(Statement, Occurrence, Attribute, Value);

(Entity, Statement, Occurrence, Attribute);  
(Role, Entity, Statement, Occurrence);  
(Class, Role, Entity, Statement);  
(Flow, Class, Role, Entity);  
(Behavior, Flow, Class, Role);

Class intension / extension (Context / Value).

**5.2.: Meta Resources**

Meta Model Context hierarchy classes:

Classes: Layers monads. Class hierarchy. Inputs resolves from wrapper containers to next layer occurrences (map forward), occurrences contexts collects matching result graph (reduce backwards).

ID<ID> : Reified matching URIs

Transform<ID> : Range

Mapping<Transform>

Template<Mapping> : Domain

Augmentation<Template>

Resource<Augmentation>

Role<Resource> : CSPO Role

Statement<Role> : CSPO Quad

Model<Statement> : Set of Statements

Meta Resource / Model: encode Model, URIs / Layers / Contexts / Facets / Levels / Resources hierarchies. Mappings.

Meta Resource / Model: Encode Message, Template, Augmentation(s), Transforms and Mappings (Dataflow).

Meta Model: Data, Session, Interaction Levels (Message, Template, Transform, Augmentation statements). Mappings.

Model Context / Layers, Facets, Ontology levels, Meta Resources / Models mappings / reification. APIs. Levels example: Behavior / Interaction (Action, Gesture..., Flow). Upper ontologies: Action, Gesture etc. classes.

Contexts / Layers / Levels / Facets Meta Resources / Models classes / instances hiers (ontology matching / data, schema, behavior alignments). Members: URIs, Resource, Context, CSPO, Meta Resource / Model APIs.

Functional API: Message IO. Mappings.

State order (in context class hierarchies axes), comparison relations, iterations, flow, events, causal relations, units, enums, equivalence, etc.

Data order: Resource Kind hierarchies.

Schema order: Role Class hierarchies.

Interaction order: Statement Context hierarchies.

**5.3.: Meta Model**

Messages: Augmentation (performed transform), Template, Mapping (possible transform), Transform.

Monads / Functors examples (order: comparables / upper / dimensional ontologies):

Kind<SubjectClass<Resource>, Set<PredicateClass<Resource>>>;

Example: Subject / Predicate(s), others (quads prev / next relation).

Metaclass<ObjectClass<Resource>, Set<OccurrenceClass<Resource>>>;

Example: Object / Statement(s), others (layers parent / child occurrences relation).

Class<ObjectClass<Resource>, Set<ContextClass<Resource>>>;

Example: Resource / Kind(s), others (class definition relations: extension / intension, layers parent occurrences prev relation).

Reactive Uniform Component API: Context reactive instances: network addressable / operable (pub / sub streams). Wrapper API.

Addressing: Encoding. Network URLs, Semantic URNs. Naming, Index, Registry operations.

Dataflow Routes: Context Signatures. Forms. Bus. Addressing dispatch resolution.

Message: IDs Statement. Specifications (CRUD).

Template Matching: Alignment / Encoding. Populate Template with Message. Map.

Augmentation Mappings: Flows (Wrapper API). Exchange.

Materialize Specification: Transform (Wrapper API). Reduce.

Classes: Layers monads. Class hierarchy. Inputs resolves from wrapper containers to next layer occurrences (map forward), occurrences contexts collects matching result graph (reduce backwards). Map / Reduce: Graph key / value / properties encoding.

ID<ID> : Reified matching URIs.

Message (encoded reified verb?)

Transform<ID> : Augmentation Range

Mapping<Transform>

Template<Mapping> : Augmentation Domain

Augmentation<Template> : Domain / range signature

Resource<Augmentation>

Role<Resource> : CSPO Role

Statement<Role> : CSPO Quad

Model<Statement> : Set of Statements Role Resource Occurrences (Kind?)

Messages: Augmentation (performed transform, Flow), Template, Mapping (possible transform, Form), Transform. Dialog.

Model Reactive I/O:

Model forward (map inputs): aggregate inputs into reified layers contexts instances (Model Meta Resources reification).

Augmentation: populate / perform Flows. Aggregate, Align, Activate (over mapped inputs). Mapping Template Transform algorithms / services encoding in Statement plus Meta Resources.

Model backwards (reduce outputs): collect occurrences graph (matching signatures contexts from Model layer to IDs).

ContextClass<OccurrenceClass> : Layer (IDs). Attributes, Values.

Events: Message Monads (IDs hierarchy instances), Functors (layers classes instances reifying model classes / domain instances from facets / levels). Flow: Augmentation materialized Transform. Form: Mapping possible Transforms. Browse / Apply (generic forms, flows?, Wrapper API).

Layers:

Resource?

(ID, ID, ID, ID); Message (encode reified verb?)

(Transform, ID, ID, ID);

(Mapping, Transform, ID, ID);

(Template, Mapping, Transform, ID);

(Augmentation, Template, Mapping, Transform);

(Resource, Augmentation, Template, Mapping);

(Role, Resource, Augmentation, Template);

(Statement, Role, Resource, Augmentation);

(Model, Statement, Role, Resource); Resource Occurrence in Model (Kind? Kind Role type, Kind hierarchies.)

Functional layers?

Resources (reactive entity: quad / ID):

Layers Context types: reified Resource quads instances (Meta Model Kinds). Resource quad wrapping: signatures bindings.

Context layers instances: Meta Model Kinds hierarchy. Resource quad wrapping: signatures bindings.

Meta Model Kinds hierarchy (super / sub Resource class / kinds rels: super: ctx kind, sub: obj kind). Reified model entities: layers super types,

Uniform Reactive Resource Quads Wrapper API (Resource / IDs / Message / Layers): Metaclass (P) / Class (C) / Instance (O) / Occurrence (S) CSPO Resource roles / rels members, monads / transforms. Events domain / range.

Reactive component: Resource, pub / sub (endpoints APIs) for wrapping signature bindings (layers). Events domain / range.

Aggregations: Subject / Attributes, Attributes / Object, Subject / Object (Kinds). API for Functional layers interaction / composition.

Enable uniform treatment of Resources for layers aggregation / augmentation, etc.

Message / Specifications (Mappings Forms / Flows). Encoding (Specification, Form, Flow) of Mapping Transforms.

Transform. Compare: common upper types.

Encoding: nested shapes of recursive cuads (till primitives). Patterns / expressions: wildcards, variables, placeholders:

[[123, 456, \_b, $a][\_b][\*][$a]]

Resource Component:

reactive resolution / instantiation events matching conditions.

Resource class / component kind:

Members. Relations: Graph quad layers bindings (DOM). Previous, next, parent, child (order: class hier relations), Resource (instance), Role (metaclass), Statement (occurrence), Kind (class). Resource Monads. Eval rels axis: instance. Functors: ID Monads rels traversal.

(Context, Occurrence, Attribute, Value);

Events API. DOM. Monads. Functors (domain / range). Relations / traversal. Events. Encoded in Meta Model (Message Monad).

Encoding: signatures / bindings. Representation query / traversal / transform. Class / instance Functor / Monads relations Dataflow.

Resolution / instantiation: Resource & Resource members / graph quad layers bindings.

Statements: Backend. Encoding. Addressing. IDs. Sync Functional Object Model. Services. Reified Object Model. Onto aligns: upper / dimensional. Messages I/O: IDs matching / alignments.

Functional Object Model:

Events (Message I/O) conditions matching (resolution / instantiation / bindings).

Objects: componed of aggregation of monad resources of a reified ID.

Flows: (Resource (Role (Statement (Kind))));

Forms: (Template (Mapping (Transform (Augmentation))));

Object: (Value (Attribute (Occurrence (Context))));

Members, relations, endpoints (API).

Message I/O: Dispatch according signatures bindings. Augmentation events (Functors) Kind streams.

Message<Monad<ID>>: Objects matching Message encoded structure (value, attribute, occurrence, context, class / instance, rels). Resolve, instantiate: perform Message logic (encoded in Model monads / Meta Resources). Example: Map Reduce. Return Dialog Graph Message.

ID

Message<Monad<ID>>

4 Transform: Range Flow

3 Mapping: Bind Specification

2 Template: Domain Form

1 Augmentation: Verb / Event. Browse rels

4 Resource

3 Role

2 Statement

1 Kind: Event streams

(...Model, Functional? Class, Entity, Flow, Behavior)

OGM: Object Graph Mapper. REST Facade. Domains activation.

Meta Model: encode Layers, Contexts, Kind / Roles hierarchies (subject, context, occurrence, roles, atributes, values / metaclass, class, instance relations / meta resources).

Augmentation: Described in Meta Model. Encode Message, Template, Augmentation and Transforms roles (Meta Resources). Data, Session, Interaction Levels (Message, Template, Transform, Augmentation statements declaration realization).

Augmentation. Aggregation Meta Model: Describe layers contexts compositions. Alignment Meta Model: Describe augmented attributes (by kinds clustering). Activation Metamodel: Describe Kinds / Roles activation (by attributes aggregations).

Encoding. Addressing (contents, signatures, contexts). Events publish / subscribe. Dynamic subscriptions / bindings. Subscription, reactive Meta Resource(s). Message flow mechanism: from Model to base layers.

Context: Resource / Message Monad Events: Augmentations. Mapping: Endpoint. Events: Implement Message / Resource / URIs Protocols.

Monadic wrapper for which Augmentation (Functor Events) are declared into Interaction Model. Model(s) themselves are Augmentation(s). Augmentation Statement Context Kind defines Event “signature”: Resource input / output Event domain / range. Output from an Event application (Transform) may feedback Model triggering further events (Dataflow). Augmentation Template, Mapping and Transform may behave as placeholder for Dataflow rendering of Meta Models.

Model Resources react to events according Message matching event “signature”.

Message (Subject : Data level)

Template (Context / domain : Session level)

Augmentation (Occurrence, declarative / service Resources: functors. Interaction level)

Transform (Role / range: Kind transform matches. Session level). Resulting Message Attribute / Value roles populated.

Meta Model encodes:

Meta Resource class / instance patterns.

Participation: Subject in Occurrence.

Role: Participation for Subject.

Kind / Context hierarchies.

Mappings: Facets (Models / Contexts declarations) by Meta Resource statements in Meta Model. Mappings renders Model(s) contents statements (layers) by Context Augmentations.

Augmentations defined as declarative Mappings in Meta Model encoding Context (layer) inputs matching signatures and augments current / previous layer emmiting mapping transforms. Context : Functor. Participation wraps Context / Resource.

Aggregation (Augmentation): Apply each Context (layer) Functor on inputs (from input layer) and emits Transform, matching corresponding (next) layer. Next layer Context and SPO according functional mapping declared by Meta Resource types on augmented layer.

Alignment (Augmentation): ToDo.

Activation (Augmentation): ToDo.

**5.3.1.: Facets**

Models have “Facets” which renders the different ways Model data / schema / behavior could be regarded as and used for different purposes, from application development to Business Intelligence and Ontology Matching.

Facets are models implemented the same way other models are with Model Resource Contexts and layers and from the same data. Each Facet implements its own Resource URI wrapper (same URIs, ontology matching, provenance of aligned URIs, Facet pivoting). Then, each Facet has its own Model Context Resource class hierarchy having Augmentation / Dataflow functors as Model Resource(s) does.

Facets: Context to Model Mappings. Data / schema / behavior class / instances views (aggregation) APIs. Model Meta Model layer aggregations.

Several types of Model(s) exists: Facets, each one preserving this layered structure. Model Facets have corresponding Layers and those layers are populated by corresponding Data, Schema, Behavior conforming Ontology Levels for each Facet. Facets abstract Model(s) inputs regarding this aspects: Source (Functional) Data, Semiotic and Dimensional Model Facets.

Facets are also populated in what are called Ontology Levels, which are Facet data, schema, behavior statements aggregated from feedback from the data, schema and behavior corresponding instance layers of the Facet Models themselves again into the input layer thus allowing for further describe upper ontology abstractions. These upper abstraction may be grouped into: Backend / Source (Data : plain inputs), Grammar / Session / Context (Schema : schema layer feedback inputs) and Interaction (Behavior : behavior layer feedback inputs).

**5.3.1.1.: Functional Facet**

Facet Aggregated Quad Statement Layers:

Layers are implemented as an RDF Quads hierarchy aggregating each one on top of another. The idea is that aggregating Data according some criteria one could enable us to infer the Schema that those Data belongs to and that aggregating Schema and Data one could enable us to infer the Behavior (operations) that correspond to the Data manipulation in that corresponding Behavior layer class / instance.

The idea is to infer Schema (classes and instance of classes / relations) operating over Data layer. Then, by aggregating Data and Schema, infer Behavior (classes and instances of operations / functions). This Aggregation, together with Alignment and type Activation mechanisms comprehend the Model core Augmentations.

(Statement, Role, Resource, Augmentation);

(Entity, Statement, Role, Resource);

(Kind, Entity, Statement, Role);

(Class, Kind, Entity, Statement);

(Flow, Class, Kind, Entity);

(Behavior, Flow, Class, Kind);

**5.3.1.2.: Semiotic Facet**

Facet Aggregated Quad Statement Layers:

(Context, Sign, Concept, Object);

Object as Sign: Concepts represents attributes (DOM / OGM). Ontology Matching (shapes).

Semantic / Semiotic Facet:

Resource : Functional URI wrapper.

(Context : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);

(Attributes, Occurrence, Attribute, Value);

(Object, Attributes, Occurrence, Attribute);  
(Concept, Object, Attributes, Occurrence);  
(Sign, Concept, Object, Attributes);  
(Context, Sign, Concept, Object);  
(Interaction, Context, Sign, Concept);

(Model, Interaction, Context, Sign);

**5.3.1.3.: Dimensional Facet**

Facet Aggregated Quad Statement Layers:

(Value, Previous, Distance, Next);

(Measure, Value, Previous, Distance);

(Unit, Measure, Value, Previous);

(Dimension, Unit, Measure, Value);

(Concept, Dimension, Unit, Measure);

(Context : Resource, Occurrence : Resource, Attribute : Resource, Value : Resource);

(Properties, Occurrence, Attribute, Value); Data (Properties: distance / facts).

(Value, Properties, Occurrence, Attribute); Info (Properties distance between Occurrence / previous and Occurrence / next).  
(Measure, Value, Properties, Occurrence); Knowledge.  
(Unit, Measure, Value, Properties);  
(Dimension, Unit, Measure, Value);  
(Concept, Dimension, Unit, Measure);

(Model, Concept, Dimension, Unit);

Dimensional alignment / aggregation layers (lower resource alignment layers):

(Value, Distance, Prev, Next : in Units); (Measure, Value...) (Unit, Measure, Value,...); (Resource, Unit, Measure, Value); Marriage event example.

Model Contexts: Meta Resources / Contexts hierarchies. Models:

Data: Source / Interaction, Schema: Encoding / Grammar, Behavior: Dimensional / Measures (marriage).

Example:

(Value, Previous, Distance, Next); Person, Single, Marriage, Married; Man, Single, Marriage, Husband; Woman, Single, Marriage, Wife.

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);

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.

Encoding / Dimensional example: role in context. X is Y for Z in W.

(W (Z (X (Y))));

(Marriage (Role (Man (Husband))));

(Hour (Minute (1 (60))));

Encoding, Dimensional, Meta Model. Units. Events. Order. Relations. Comparison. Input layers. Augmentation.

Events metamodel (TBD):

**5.3.2.: Layer Levels**

Models / Meta Model Levels: reify models Contexts hierarchies into IDs layer. For each model layer into different hierarchies reify an ID layer quad statement corresponding to those layer Context and perform corresponding layer(s) aggregations. Inferred “upper” ontologies. Mappings population / resolution.

Models have “Ontology” Levels. Levels are Layers (of the Model) which are feed into its input Layer with (instance) Statements aggregated from initial input data (Data Level) aggregated into subsequent layers. Schema Level instances feeds the Model input conforming a Session (context / grammars) ontology Level. Then, behavior Level instances feeds the Model input conforming an Interaction (behavior) ontology Level.

Examples: Source, Session, Interaction declarative application protocol use case upper ontology levels (Action… Gesture, etc).

**6.: ID: Context Reactive Abstraction**

Model IDs. Service (Connector / Client) IDs. Addressing reactive abstraction.

Context<ID>; Signature route. Reactive producer / observer.

Mapping<Context<ID>, Context<ID>> : Context<ID>;

Mapping<Template : Person (S), Transform : Address (O)> (P) : Context<ID> (C) : Augmentation;

Mappings Encoding: parse Template in context of Transform. Augmentation: materialized result Resource (query / assertion).

Meta Model: IDs to Context hierarchy Mappings.

Meta Model levels reification populates / resolves Mappings.

Queries / Assertions: Domain driven and Core Augmentation Messages: Model Message layout (Flows browsable API) defined in models levels reifications (grammars / layers / facets / levels).

Model Message layout Dataflow: Mapping routes, Templates, Transforms signatures matching (bindings).

Model Message layout Augmentation: Message input transform / alignment (raw quads: ontology matching / match Model patterns). Message<Context> : apply Dataflow transforms. Emit resulting Message (dialog / feedback).

**7.: Encoding**

Model Encoding: Property graph. Properties (prefix codes, key / value, reification). Sets, groups, categories. Functors applications: Transforms as graph navigation / browse. Template Message parsing (grammar, verbs, state flow). Contextual Quad Context ID: ID according occurrence in Statement context (normalized forms). Occurrence Context IDs indices / mappings.

Encoding, IDs: magic numbers (MIME types : Context Kinds), metaclass, class, instance, context, CSPO, etc. relations "contextual slots" for IDs. Resource resolution, Operation (primes, encoded lattice, slots context relations) factors in Meta Model relations. Encode order, hierarchies, temporal, causal (reified), containment, etc. relations into IDs encoding. Ontology matching: encoded IDs roles in context aggregation / learning.

IDs: Addressing / Encoding. Semantic (signature, contents, context) resolvable / discoverable identifiers.

Ontology Matching:

Ontology Matching: IDs, Addressing, Encoding. Functional, Semiotic, Dimensional (Facets). Layers. Levels. Meta Resource / Model. Sets. Value as occurrence of attribute. metaclass / class / instance IDs.

Ontology Matching: Encode: order, iteration, flows, units, relations, events, enums, etc.

Messages CRUD / Invocation semantics. Dialog. Prompts.

Encoding: Cons lists. Trees. Huffman / Prefix codes. RDF List serialization. Meta Resources / Models declarative Encoding, Addressing, Mappings, Transforms (Immutable sequences, dataflow Mapping: Template / Augmentation / Transform functional streams).

Kinds, Signatures. Contents. Contextual metadata. Sets (bitstring cuads). Lattices.

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors (Augmentation) behavior encoded in statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

Interaction Model (Interaction Level):

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

Order / comparisons: tree representation ordered by Context Role class hierarchy, instances hierarchies and aggregation hierarchies. Resources order (IDs). Statements order (Statement IDs). Comparison criteria (choose relevant IDs). ToDo.

Semantic resolution: Query Resource(s) satisfying “criteria” (i.e.: Object(s) for predicate) IDs by IDs resolution pattern:

Query Resources by role in context.

Query Resources by attributes / values.

Query Resources by identity / type.

Context Kind: Functional stream of Context Statements (Occurrences).

Subject Kind: Functional stream of Subject Statements (Occurrences).

Predicate Kind: Functional stream of Predicate Statements (Occurrences).

Object Kind: Functional stream of Object Statements (Occurrences).

Message - Model - Template (data) - Augmentation (functor) - Transform (interaction) - Model - Message.

Ontology Matching:

Ontology Matching: IDs, Addressing, Encoding. Functional, Semiotic, Dimensional (Facets). Layers. Levels. Meta Resource / Model. Sets. Value as occurrence of attribute. metaclass / class / instance IDs.

Ontology Matching: Encode: order, iteration, flows, units, relations, events, enums, etc.

Messages CRUD / Invocation semantics. Dialog. Prompts.

Encoding: Cons lists. Trees. Huffman / Prefix codes. RDF List serialization. Meta Resources / Models declarative Encoding, Addressing, Mappings, Transforms (Immutable sequences, dataflow Mapping: Template / Augmentation / Transform functional streams).

Kinds, Signatures. Contents. Contextual metadata. Sets (bitstring cuads). Lattices.

(C (S (P (O, Nil))));

(C2 (C (S (P, Nil)));

Encoding / Dimensional example: role in context. X is Y for Z in W.

(W (Z (X (Y))));

(Marriage (Role (Man (Husband))));

(Hour (Minute (1 (60))));

Encoding, Dimensional, Meta Model. Units. Events. Order. Relations. Comparison. Input layers. Augmentation.

Meta Model:

IDs:

URI(s);

OntResource; Merged URI(s) wrapper.

OntResource hierarchy: layers statement contexts. Facets DOM, Actor / Role.

Resource (OntResource Context Roles hierarchies Monad wrapper);

Statement : Resource quad, Resource.

Message (Resource Monad wrapper); Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

IDs:

A: ID

B: Transform

C: Mapping

D: Template

E: Augmentation

F: Model

G: OntResource

H: CSPO Role

I: Statement

J: Kind

K: Class

L: Context

State Facet / Layer / Level / Augmentation / Model Resource Mappings.

Meta Resources layers contexts class hierarchy:

Context : Class : Kind : Statement : Role : Resource : Model : Augmentation : Template : Mapping : Transform : ID;

Meta Resources layers Reification: Context : Class : Kind : Statement : Role : Resource : Model : Augmentation : Template : Mapping : Transform as ID Context Statement (ID statements for each context layer). Model Levels (Facets / Levels). Aggregate reifications.

Meta Model:

A: (ID, ID, ID, ID);

B: (Transform, ID, ID, ID);

C: (Mapping, Transform, ID, ID);

D: (Template, Mapping, Transform, ID);

E: (Augmentation, Template, Mapping, Transform);

F: (Model, Augmentation, Template, Mapping);

G: (Resource, Model, Augmentation, Template);

H: (Role, Resource, Model, Augmentation);

I: (Statement, Role, Resource, Model);

J: (Kind, Statement, Role, Resource); Data (Resource Kind).

K: (Class, Kind, Statement, Role); Schema (Role Class)

L: (Context, Class, Kind, Statement); Interaction (Statement Context).

ID: (L (K (J (I (H (G (F (E (D (C (B (A, Nil))))))))))));

Cons lists. Binary Trees. Huffman / Prefix codes. RDF List serialization. Meta Resources / Models declarative statements Encoding, Addressing.

Kinds, Signatures. Contents. Contextual metadata.Lattices. Roles.Sets (bitstring cuads). Definitions (elements). Operations. Rules. Categories. Groups.

ID: Augmentation occurrences metadata in Statement contexts;

Statement: (ID (ID (ID (ID, Nil))));

(C (S (P (O, Nil))));

IDs, Meta Model, Interaction Model (Level), Session (Level), Backend (Level), Facets features:

Augmentations: Interaction Model Mappings execution / persistence / retrieval. Reactive model via representation of IDs: Mappings (signatures) dataflow inferred Augmentations.

Persistence: (activation / passivation): IDs / Meta Model / Facets from Interaction Model events (Messages) from Node IO. Interaction Model: Main Model(s) Message IO.

Order / comparisons: tree representation ordered by Context Role class hierarchy, instances hierarchies and aggregation hierarchies. Resources order (IDs). Statements order (Statement IDs). Comparison criteria (choose relevant IDs). ToDo.

Semantic resolution: Query Resource(s) satisfying “criteria” (i.e.: Object(s) for predicate) IDs by IDs resolution pattern:

Query Resources by role in context.

Query Resources by attributes / values.

Query Resources by identity / type.

Message (Resource Monad wrapper) : Statement; Request / Response Encoding.

Template / Transform (Message blueprints) domain / range : Message.

Augmentation : Functor.

Mappings: Immutable Resources Message based Augmentation bindings. Dataflow subscription routes: Signatures / CKs (Augmentation(s) functional streams).

Subject Kind: Subjects stream. Object Kind: Objects stream.

Encoding: Template Message augmentation (inputs).

Encoding: Declarative functors (Augmentation) behavior encoded in statements. Mappings (subscription / routes).

Encoding: Transform Message augmentation (outputs).

Interaction Model (Interaction Level):

(Augmentation, Template, Mapping, Transform);

Dataflow: Order, Flows (Mappings, hierarchies).

Interaction Model: aggregated Meta Model interactions (performed / inferred / possible) declared Models events (saga pattern).

Messages: Saga Activation. Interaction Model (Meta Model). Aggregated (Interaction) Meta Model interactions (performed / inferred / possible) emitted as Model event Messages (Saga pattern). Mappings.

**8.: Signatures**

CSPO Context Kind (Statement Subject Kind + Object Kind). Context Dataflow domain / range (Context as reactive streams producer / consumer).

**9.: Routes / Dataflow**

Routes: Dataflow pub / sub bindings between matching signatures.

Core Model and Domain driven Message flow layout (Mappings).

**10.: Event Bus / Messages / Addressing**

Dispatch Event into Dataflow Route.

Messages:

Monadic Functional Statement (Resource) wrapper.

Messages: Message semantics (Augmentation: Verbs, CRUD, Behavior) according Message structure / pattern (dialog / prompts).

Augmentation result: Message / interaction layer matching / populated Transform Statement. Template, Mapping, Transform Augmentation Meta Resources.

Messages: Dataflow Template matches signatures (Session level, enrichs Message with Model / Dialog prompts / contents). Augmentation Functor applied over Message contents (Interaction level). Transform matching output signature emits (Session level, populated / prompts) output Message.

Persistence:

Messages: Events IO / Persistence: Saga Activation / Passivation populating Node local Quad store / persisting peers via DIDs ([ont.io](http://ont.io)) semantic (resolvable / discoverable) identifiers.

Interaction Model: aggregated Meta Model interactions (performed / inferred / possible) declared Models events (saga pattern).

Augmentations: Interaction Model Mappings execution / persistence / retrieval. Reactive model via representation of IDs: Mappings (signatures) dataflow inferred Augmentations.

Persistence: (activation / passivation): IDs / Meta Model / Facets from Interaction Model events (Messages) from Node IO. Interaction Model: Main Model(s) Message IO.

Messages: Saga Activation. Interaction Model (Meta Model). Aggregated (Interaction) Meta Model interactions (performed / inferred / possible) emitted as Model event Messages (Saga pattern). Mappings.

Saga Activation / Passivation populating Node local Quad store / persisting peers via DIDs ([ont.io](http://ont.io)) semantic (resolvable / discoverable) identifiers.

Messages: Saga Activation. Interaction Model (Meta Model). Aggregated (Interaction) Meta Model interactions (performed / inferred / possible) emitted as Model event Messages (Saga pattern). Mappings.

Messages: Saga Passivation. Model layers data routed by Mappings as event Message into (Interaction) Meta Model. Message inputs: Models. Mappings. Populate.

Messages: Dataflow. Subscriptions. Reactive Model. Dynamic subscriptions / bindings. Events publish / subscribe between Model Resource. Mappings.

Messages: Saga Activation. Interaction Model (Meta Model). Aggregated (Interaction) Meta Model interactions (performed / inferred / possible) emitted as Model event Messages (Saga pattern). Mappings.

Messages: Saga Passivation. Model layers data routed by Mappings as event Message into (Interaction) Meta Model. Message inputs: Models. Mappings. Populate.

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);

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.

**11.: Model I/O**

Dialog Protocol:

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

**11.1.: Mappings: Events Transforms Declarations**

Interaction Model declares Events (Augmentations) which have a functional Mapping between its domain (Template) and range (Transform). An Augmentation Context Kind correspond to this Mapping “signature”. Dataflow binds input Message(s) to domain Template by pattern matching and resolving any input Message references (Addressing).

Outputs are resolved by pattern matching with Transform, Message and existing Model data. Augmentations may play the role of “placeholder” Resource(s) which are bound to context aware Augmentations thus rendering Transforms into Model entities (including Mapping Augmentations themselves).

**11.2.: Core Augmentation Mappings**

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 possible into distributed peers which exposes protocols / APIs for knowledge driven hypermedia applications.

Extension / Augmentation: BI / EAI. Smart dashboards / reports / workflow / process / activity components. Activable smart indicators / components (predict / execute). Declarative Model interpretation into abstract application models. Rendering (Gestures ontology).

**11.2.1.: Aggregation**

Aggregation: infer roles in contexts: regression (Person class in Employment interaction : Developer role).

**11.2.2.: Alignment**

Alignment (infer attributes / relations): clustering (from multiple occurrences of same entity in diverse data sources).

**11.2.3.: Activation**

Aggregation: infer roles in contexts: regression (Person class in Employment interaction : Developer role).

**11.3.: Augmentation: Events Mappings Realizations**

Functors: Augmentation declaration: Meta Model definitions (Context class / instances). Message: dataflow matches Template signatures: interactions. Apply Augmentation Functors over Message contents (interactions enrich Message with Models contents: ontology matching / Levels / Facets). Materialize / emit dialog / prompts Message (enrich Message from Models / reactive IO events).

Augmentation: Context / Functors. Message Resource(s) / Meta Resource(s) (nested / wrapped) elements determines flow Template Transform results / behaviors (CRUD, Functor invocations). Message IO performs Augmentations. Ontology levels resolution (Templates / Transforms / Augmentatiom levels: matching patterns / dialog prompts in Ontology levels).

Interactions declarations: signature definitions (Template / Transform contexts). Interaction instances: addressable exchanges (Augmentations, Message, Model context / Mapping bindings / matchings / performances). Meta Model / Levels event driven Model Augmentation.

Interactions declarations: signature definitions (Template / Transform contexts). Interaction instances: Exchanges (Augmentations, Message, Model context / Mapping bindings / matchings / performance). Contexts / Exchanges: Meta Model / Levels event driven source Augmentation events declarations (populating Facets / Layers / Levels).

**12.: Backend**

**12.1.: Model Containers**

**12.1.1.: Services / Protocols**

Layers (session, dialog, etc.). Node, Peer, Client, Connector, etc. Reactive / Event Driven. REST HATEOAS.

Protocol (deployment):

Functional "Dialog" Augmentation Semantics Protocol (Dataflow Message).

Applications:

Hypermedia Dataflow Activation (reactive / event driven knowledge based contents). Dataflow layers.

Distributed: Consistency. Inference of distributed state. Event sourcing. Trust. Reconciliation.

Connected application sources (backends: EAI / ESB) and declaratively stated application models.

Index, Naming, Registry.

**12.2.1.: Models Declarative Encoding**

**12.2.2.: Functional APIs**

Index, Naming, Registry.

MapReduce.

**12.3.: Persistence**

Blockchain DIDs.

MapReduce (Encoding).

**12.4.: Connectors**

Tryton

GNU Health

Apache Metamodel

JBoss Teiid

JBoss Drools / JBPM

OData

R2RQ / R2RML

SPARQL

ISO 15926 / ISO Topic Maps

DCI / MVC: JDBC / OGM / ORM / JCA / Activation JAF / Process Flows (state)

Declarative hypermedia: REST / HAL / HATEOAS

Declarative hypermedia: SOAP / WSDL

CMS / Wiki (API / Protocol / DAV). Docs. Forms (Docs Flows)