* Schedule. Tasks.
* Notes : Topics.
* Topics: P2P, Blockchain, DIDs. Ont DIDs.
* Topics: RDF(S), OWL, SPARQL, ShEx, SHACL. Notation3, Rules.
* Topics: ISO, TMDM, TMRM.
* Topics: Math.
* Topics: HAL / OData / GraphQL / Protocols.
* Topics: Vert.x / P2P / JXTA
* Topics: RxJava / Java 8 / Functional / Streams.
* Topics: Functional JS / OO JS / NodeJS
* Documents: Topics. Tools. Specification.
* P2P DIDs (Ont DIDs).
* Proof of work: Encoding (equation) resolution.
* Reactive abstractions over P2P layer. Producer, Consumer, Processor, Message. P2P Browser. Platform. Container (devices layer).
* Deployment. Interfaces (endpoints: client / server, APIs for reactive abstractions). Network: interfaces, addressing, messages, protocols, routes implementing services / augmentations (encodings).
* Distributed CAM. Encoded in devices. Event sourcing (sync: indirect knowledge predictions / inferences). Augmentation: assertions / rules.
* Interfaces / QA / Wizards / Queries: which day of which month was the Wednesday before of last year Easter and how many patients where diagnosed of flu that day and whose of them where smokers and whose of them lived near some region being workers of some factory.
* P2P Browser: search session history. Referrers. Purpose steps. Role navigation bookmarks. User interface (declaratively domain driven protocol / queries / assertions).
* Content type activation: smart semantic overlay endpoints, metadata, description, addressing, linking, augmentation. Application and filesystem augmentation. Syndicated file, application, content, behavior manager. P2P Browser. Universal Plugins (Protocols, APIs, implementations): Assisted Contexts P2P Browser. Inter domains oriented.
* WebDAV virtual filesystem plus index, registry and naming description / linking / addressing metadata services on device for P2P Browser storage / retrieval / activation. Ont DIDs services. DID Endpoints: WebDAV, OData, HAL, SPARQL, other protocols (WS, RPC, etc.).
* Context Resource produces occurrence / attribute / value events (aggregated by occurrence). Context URI events bus.
* Occurrence Resource consumes resource occurring events (in context, with attributes / values. Occurring URI events bus.
* Message: transform T > R (subclasses of URI: content types, APIs) available verbs / flows / navigation (state). Dynamic Resource monad over URIs.
* Message: Metamodel Resource / URI instances / API wrapped as reified Message. DOM. Object navigation.
* Activation Message: onSubject<Statement, Kind> materializes SubjectKind. Idem PredicateKind, ObjectKind.
* Aggregation Message: onLayer<Statement, Statement> materializes next layer Statement.
* Augmentation Message: ID / Classify, Attributes, Context Role.
* Contexts (dataflow): possible message flows invoked as available inputs / materialized results matches transforms signatures.
* Messages: dispatch. Processors. Event bus topics for each model layer types and for each resource type URI instance.
* Queries: layers traversal / aggregation (streams). Person as Employee in Sale of Offers.
* P2P Browser: Addressing, linking and annotation / embedding. Items: subjects occurrences in roles in contexts: Documents / resources semantic linking. Workflow session abstraction of resources in roles in learned / inferred domains (applications). Content types, addressing, representation (i.e.: mail / chat conversation URI addressed / linked as project / document cause role, person in picture occurrence in marriage event role, etc.). P2P Browser rendering declarative state flows / contexts / interactions embeddings addressable / embedded resources in roles in contexts.
* Search, find and browse semantically. Copy and paste any meaningfully labeled addressed resource into any other context resource. Discover intelligent insights from linked knowledge bases rendered together with the resources that originated them while navigating with an enhanced browser which allows for conceptual based relations and dimensional reasoning traversal.
* Connect any addressable content type representation into document (semantic resource) embeddings playing roles: purchase, invoice. Resource annotations. Addressing and type / representation handlers as browser plugins. Custom protocol adapters (example: mail, chat conversations playing role in interactions; picture: scanned receipt). Meta browser involving session workflows. Annotations (subject / occurrence: picture / event; paragraph mentions interview: semantically augmented link to audio / video of subject).
* Basic inference (resource augmentation: activation, aggregation, alignment models).
* Class (type) / ID (instance) inference.
* Attributes / Links inference.
* Context / Role inference.
* Message driven reactive P2P architecture. Synchronization. Event sourcing (distributed consistency).
* Implementation: Health Care domain. Functionalities such as kinds of ERP, CRM, SCM (B2B / B2C) for health: HMS (Health Management System) The Browser provided will act as an overlay (of integrated resources). Example linked resources (records): symptoms, measures, interviews (previous QA forms), analysis / trials, results, diagnoses. Clinical history. Integration: peer / node, cloud. Metamodels. BI / Big Data. Rules. Planner. Business Process Management. Resources allocation / logistics (optimize resources allocation). Workflows (ontology learnt).
* Implementation: Runtime. Architecture. Components. Patterns. Models. Messages. Augmentation. Events dispatch. Message aggregation / instantiation / resolution / application. Backends. Services.
* Resources Inputs / Outputs: Augmentation, Protocol, Browser. Message resolution / application.
* Component: Models (data). Source: augmented input statements. APIs (Model).
* Component: Messages (contexts). Source: augmented models templates. APIs (Model).
* Component: Transforms (interactions). Source: input statement case matching Message inputs. Returns / materialize results. APIs (Model).
* Core Model API: Augmented (Aligned, Activated, Aggregated inputs matching context messages) IO. Resource MetaGraph. Dimensional model. Grammar. Model repository. Backend. API.
* Model, URI, Resource, Statement, Kind.
* Message: Context Model API. Input statements: Model Grammar. Augmented IO by interaction transforms of applied matching Message with model statements inputs. API.
* Transform: Interaction Model API. Input statements: Message application results. Functional application of Message over Resource. Input statements: Resource over which applicate / augment with matching Message. API.
* 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. Materialize Transform. Results.
* Services: distributed addressing / resolution, reactive distrbuted event bus: streams / events, index, naming, registry.
* Modal logic. Sets. Categories. Groups. Formal spec (data / reference models).
* Data structure
* Instance / behavior
* Wrapper 'monadic' domain: dynamic wrapper type (set / groups).
* Set Membership / Set Operations 'dynamic' domain.
* Dynamic dataflow, operation concatenation, matching (signatures). Injection (references resolution / roles), arrangement in contexts (embeddings: role / rel between members arrangement, navigation, resolution, operations: dynamic monadic types). Dynamic DOM (dimensional DCI, data / info / knowledge 'contexts': signatures refs resolution; sign / concept / object contexts: resolve elements / operators).
* Wrapper types (Token):
* Resource: dynamic type wrapper of Token (occurrence of Token in Resource). Class, Metaclass, Roles. Tensor like internal representation.
* Resource (token, C, S, P, O Resources: mappings / tensor like CSPO dimensional arrays). One instance for each context occurring in subjects with attributes / values.
* Dimensional DCI (data, schema, behavior)
* Positional roles wrapper types (Resource interfaces):
* Context (Sign, Concept) : Object
* Sign (Context, Concept) : Object
* Concept (Context, Sign) : Object
* Object (Context, Sign) : Concept
* Statements (object):
* (Context, Sign, Concept, Object)
* Statement (object) layer (DCI) type: wrapper types in positional roles.
* Layers: dimensional, data, schema, behavior.
* Dataflow: objects (statements) mappings.
* Mappings (retrieve existing or create new: functional):
* Context<T, V>(Context<T, V>) : Context<T, V> (assert data: price, infer info: variation new 'column', perform knowledge: assign roles).
* Mappings: 'forms’ serialization code / data format (homoiconic LISP like). (TID:VID (TID:VID (TID:VID))) assertion / query. Parses 'monadically’ functionally into Resources manipulations (activations). RDF triple serialization (SPO), expression resolves into Context Resources.
* Dataflow: internal Resource representations using Tensor like datatypes.
* Dataflow: statement (forms) assertion fires mappings resolutions (align / augment: infer / resolve created / retrieved references old / new statements; infer sameAs / rels / ctxs: sets, synsets, embeddings, categories, groups).
* Dataflow: Factory / builder API (Tokens / Resources).
* Dataflow: Facade (reactive inputs / outputs streams).
* Dataflow: Facade (query, browse, index).
* API: Services.
* Frontend / API: upload CSV / XML / JSON (service / form).
* Frontend / API: ontology browser, (faceted) search, index, export formats (queries / sync). Dynamic forms generation (behavior / schema / data).
* Frontend / API: batch / streaming (services) IO adapters / mappers (functional / streams) from / to CSV, XML, SQL, JSON, SemWeb, etc. to internal mappings 'forms’ serialization format (XSLT, DOM / SAX event driven like). Integration: aggregation and alignment (store state for batch / services administrable contexts / namespaces).
* Declaratively state 'pages’ (applications / DCI / MVC contexts use cases / flows) by declarative statement (internal mapping 'forms') of dimensional classes / instances of behavior / schema / data metamodels via de-referenceable (evaluable) resource streams. Application metamodel data 'upper’ ontology plus domain data ontology: rendering (services / clients) via XML (schema) IO / XSL template libraries. Content type 'activation’ (available 'forms’ queries / assertion according resource state flows).
* Dataflow: 'forms’ queries / assertions: context, member and 'types’ that could resolve / fit into 'activation’ arguments (arrangements).
* Set classes layers (contexts: Behavior sets parents of occurrence layers):
* Dimension
* Unit
* Measure
* Instance
* Occurrence
* Kind (roles)
* Class
* Flow
* Behavior
* Sets: Entities (Contexts) Serialized to CSPO Quads (quad position: role of entity in respective sets).
* Sets: Contexts (entities playing SPO / Kinds roles : reifiables).
* Sets: Subjects (identity alignment) : what it is.
* Sets: Predicates (links / attributes alignment) : set of relations of what it is in respect to what it is not (context objects).
* Sets: Objects (roles in contexts alignment) : complementary relation instances of subjects with what they are not (roles in rel w./ others).
* Sets: Subjects / Predicates: Object Kinds : Aggregate description of object instances in statement roles.
* Sets: Subjects / Objects: Predicate Kinds : Aggregate description of attributes / links of relation instances in statement roles.
* Sets: Predicates / Objects: Subject Kinds : Aggregate description of subject instances in statement roles.
* Integration: Distributed Consistency. Single distributed 'state'. Client / server, P2P actor / role / data (contexts: exchanges, request / response addressable interactions) validation. State 'exchanges'.
* Shapes: activation, dependency resolution. Class: possible number of instances (definition by extension / intension: digits / encoding).
* Ternary (balanced / cyclic) encoding (tensors) numbering system. Grammars / embeddings: digits / numbers (sign / concept / object). Event sourcing / blockchain / distributed consolidation (alignments) without distributed / centralized ledgers (seeds / hashs in state exchanges).
* Components (OSGi, Vert.x, JMX / JAF: signatures, activation, distributed resolution).
* Threads IO (streams, synchronization, ‘reference’ events).
* Augment alignements with 'feeds' (Solid, LOD, etc) connectors.
* Annotations (classes / individual subjects): attributes / values in occurrence contexts (facets / classes by attrs val aggregation).
* Contexts: occurrences in roles.
* Product / good / need: exchange token balance function (request / response exchange flows). Game theory.
* Classification / identification: data. Classification (sets).
* Attributes / links classes / instances: schema. Clustering.
* Roles in contexts classes / possible states / flows / instances: actual performances: behavior (activation, injection resolution). Regression (discrete encoded values for metaclass / class / instance in contexts).
* Encoding: action / pasion / state ctx quads of dereferenceable (applicable resources) IDs of IDs. Relations, predicates, functions, operations (injection resolution / signatures dataflow activation).
* Annotations / contexts schema. Reference 'events': compare alignments (class, attrs, roles) in rel to a temporal (octal) comparison.
* Cyclic ternary numbering: digits order relation a, b, c: a < b < c < a. Number (positional digits) weights calculation (preserve digits order rel in numbers).
* Number sets: elements, functions, relations, operations. Alignments (class, attrs, roles): encode as discrete (embeddings operable via grammars) values.
* Encoding: dimensional 'lines' for each order position. Lattice. Resolve encoding of dimensional data (instance), schema (class), behavior (metaclass).
* Unique IDs. Distributed / addressable exchanges: source, payload, destination 'validations'. State, versioning, alignments validations (possible state, local / remote 'hash' agreement).
* Ternary order semiotic positions: object < sign < concept < context < object.
* Ternary 'primitives’: self < this < that < the < self
* Quad (embeddings oprable): (Context, Concept, Sign, Object);
* Valid IDs: composite (positional) numbers such that order rel holds for ID in respect to its context.
* Alignments: production / validation of valid IDs (classification, attrs retrieval, ctx role regression).
* Implementation:
* Resources: Data aggregated layers statement sets / alignment upper resources. Root Resources facade (Layered Quads IO / transforms: locators, streams, events based activation).
* Metamodels: Domain aggregated layers statement sets / alignment upper resources. Root Metamodels facade. (Layered Quads IO / transforms: locators, streams, events based activation).
* Applications: Interaction aggregated layers statement sets / alignment upper resources. Root Applications facade. (Layered Quads IO / transforms: locators, streams, events based activation).
* Aggregation: compose stack of Resources, Metamodels, Applications IO over layers behavior activations (Resources behavior : Metamodels data, Metamodels behavior : Applications data). Knowledge activation aware resolution of multiple layer parents / children statements IO / aggregation / routes.
* Aggregation: Contexts facade (Layered Quads IO / transforms: locators, streams, events based activation). Browseable / CRUD of Contexts, Interactions, Domains, Data.
* Resources, Metamodel, Application : Context.
* Aggregation: (Application (Metamodel (Data))) : Context (Lang Quad).
* Networking, P2P, ternary routes.
* Dataflow: models, routes: data, operations (shapes, patterns, templates / codata: infer schema / behavior from data / dimensions information and knowledge).
* Dimensions: observations / measures of data.
* Data: instances / occurrences of contexts.
* Contexts: instances / occurrences of roles.
* Interactions: templates / contexts of roles.
* Event / Process. [Schema.org](http://schema.org).
* Dimensional DCI / instance, class, metaclass / templates: definitions / sets (intensional / extensional: representation, identifiers, encoding / sets).
* Naming URI scheme / encoding: dimensional / data / context / interaction hierarchical metaclass / class / instance (occurrence / template) hypermedia addressing. Discovery / browseable (HATEOAS / HAL: REST identifiers, resources, representations, DCI: identifiers encoded / messages). Layers: activation / domain / backends (aggregation, alignment, augmentation) encoding / algorithms.
* Exchange: addressable interaction (class / instance, DCI, dimensional, layers). Routing algorithms / encodings.
* Semantic tagging and search in browse sessions / threads rels. Paths. Discovery. Referrer / refs rels.
* Semiotics: Object: Resource, Concept: Class, Sign: Representation. Class / metaclass: occurrences / templates.
* Semiotics: Syntax: Data, Semantics: Context, Pragmatics: Interactions.
* Event: State change (dimensional measure in context axes), period. Class, metaclass, instance rels / templates. Data: Dimensional DCI, Information: Events, Knowledge: Data / Information aggregation, inferences, predictions (augmentation / activation).
* Activation: aggregation, alignment, augmentation, reasoning, rule, flow, schema, data inference / materialization (DCI / data, information, knowledge layers). Dataflow: ontology driven routes, patterns / ops streams (feedback), ontology (domain backend schema / DCI) templates, behavior (functional IO) events.
* Encoding: nodes (ops routes, pattern shapes: signatures), data (routes, pattern shapes: dimensional addressing) declaratively stated, dataflow bindings (discovery). Nodes as data, data as nodes: functional resources (dimensional DCI: extract signatures / addresses).
* Data, schema, behavior blend / merge / compose. Discover alignments, contexts, roles.
* Definitions, blending, loading. Frontend. Use cases. QA, Process assistant, systems integration (flows): purpose fulfilment (steps, roles, items: DCI / dimensional aggregation).
* Embeddings: encoding, semiotic ops (resource nodes / arcs: dataflow signatures / patterns).
* Sets: encoding, bitstring, predicates (order: comparisons). Convert 0/1: divide measure by possible values count. Ternary: comparators. Quads encoding: convert to aRGB / IPv4 address (routing ternary masks). Tensor shapes: recursive quads. Routing / dataflow: dynamic templates node (ops), arc (data) resources 'activation' signatures.
* Lamda WS: code representations. Resource IDs: (representations, verbs templates / contexts, types).
* Metamodel Resources: URIs, representation, messages (instance / occurrence of resource in context / dimension from browsing / referrer state, streams). REST HATEOAS / HAL by message schema resources metadata (browsing rels / state reified referred dimensional message DCI resources). Message resources: referrer / rels (resource / context). Resources as nodes / arcs (code / data) dataflow (discovery, signatures / template, patterns). Layers.
* IPv6: Resources, messages, routing / encoding (HTTP / REST / Protocols / P2P / data, schema, behavior routes: activation) as addresses (quads). Layers. Resources occurrences / messages addressing (dataflow node / arc signatures / template patterns, contexts). Message resources encoded in addresses. Addresses: encoded resources (messages / dataflow rels, ops, data).
* Resources / messages: encoded as addresses. Interactions / contexts / templates.
* Encoding, address: resource IP / port / index res. Paths: rels / arcs / roles / contexts / occurrences (other resources w./ paths, commands w./ templates, contexts).
* Beans serialization: declarative XML / IDL. Templates (commands / data). Serializable / externalizable (transforms). JAF / JCA. CDI / injection / bindings.
* Metamodel resources activation: aggregation, alignments, layers ontologies (backends functional wrappers: IO / custom logics). Resource activates on messages arrival (dataflow nodes / arcs signature / patterns / template routes / dispatch): graph activation (dataflow to corresponding 'listener' signatures). Response flows to declared dataflow / patterns 'listeners' (referrer / contexts) resources.
* Aggregation alignment.
* Identity alignment: Metaclass, class, instance, occurrence / template: dimensional (layers) identity alignment / entity blend / merge / data / schema / behavior match. Classification (dimensional state / layers). Data (reified).
* Attributes alignment: Clustering (dimensional state / layers). Schema (reified).
* Roles alignment: Regression (dimensional state / layers). Behavior (reified).
* Encoding: resource addresses as relative 'pointers' (template / occurrence pairs resolutions / rels routes, transform identifiers). Contexts: dimensional instances of types in roles. Composition of resource data / occurrences in resource templates (dataflow activations). Semiotics: contexts, signs, concepts, objects.
* Resource metadata: message / schema. Domains activation: aggregation / alignments. Generalizations / specializations: layers materialization in source schemas (interoperability). Resource extraction (dimensional data / schema / behavior aggregation / activation inferences) from statements (backends) IO Resources. Behavior / schema as dimensional (aggregated / reified) data.
* Dimensional behavior, schema, data aggregation / activation (inference / merge). Resources reified as dimensional behavior, schema, data.
* Input / Output statements: semiotical encoding (context, sign, concept, object); IO aggregates / de-aggregates dimensional data, schema, behavior statements (Resource activation / inference). Activations layers alignments.
* Extension: align / augment dimensionally existing data, schema, behaviors IO / sync. Add DOM entity type / property / event / invocation / services (IO handle: de-normalize IO: type, instance, attribute, value: behavior, schema, data dimensional reified DOM / backend / service: signature, invocation attrs / values in / out pairs).
* Domains: purposes. Integration (dimensional data, schema, behavior to be fulfilled). Goals / backtracking: DOM extension integration models: DOM declaratively states 'extensions’ to backends dimensional data, schema, behavior by specifying expected DOM states / events.
* Generic Virtualized Domains Metamodel Frontend: Resources activation editor. Dimensional data, schema, behavior browser / editor: aggregated, aligned, augmented entities, forms, flows (identity, attrs / rels, roles). CRUD / Flows / BI over consolidated virtualized domains sync (backends). Discovery / rendering / protocol over metamodel statements / resources messages. Generic resource protocols / services: parseables, renders into application dimensional DCI forms.
* Resource metamodel: quad roles (dimension / context, occurrence / behavior, attribute / schema, value / state / data). Aggregated into messages / metamodels (entity, schema, behavior merge).
* Statement roles (dimensional, data, schema, behavior) 'relative’ to contextual evaluations (functional, ordering, reification). Class, metaclass, instance context / occurrence templates: context hierarchy behaviors before dimensional (occurrence of) dimensional statements.
* Resource address: context address + occurrence address (addresses, recursive + attr / val address contexts).
* Resource representation: occurrence in context (addresses) attributes / values (addresses): ctx / occur attribute / value Statements (addresses).
* Address: quad (ctx:occur). Encoding.
* Address Resolution: (peter:x, emp:devel, :, :) : (peter:someDate, emp:softwareArchitect, salary:annual, 60000:USD); other attrs…
* Dimensional quads resource addresses context / occurrences / attrs / vals resolution: aggregated / inferred from context (browse state, link rels, referrer, etc.). Encoding.
* Activation: query contexts for kinds, flow occurrences, etc. resolution. Query for aggregation learning (listen possible dataflows).
* Kinds: Dimensional data, type, behavior / roles (dimensions) inference (declarative definitions / templates) by aggregation of occurrences (metaclass), attributes (class), values (instances). Sets / quads encodings (semiotical, context / occurrence / attribute / value, recursive statement / kind / SPOs).
* Resources monadic pattern:
* Class<Observable<OccurrencesSubClass>> extends SuperClass;
* OccurrencesSubClass extends Class;
* Observable: stream of Class occurrences in OccurrencesSubClass (as a CSPO member);
* Class / OccurrencesSubClass CSPO members: (C: SuperClass, S: SuperClass, P: SuperClass, O: SuperClass);
* CSPO members: SPO roles / kinds / context metadata (calculated / aggregated into kinds resources) accessible from convenience methods.
* CSPO members occurrences resolution / aggregation by metadata / context.
* Example: Statement is a subset of Resource occurring in a Kind. Resource is an URI occurring in an Statement.
* Kinds (schema layer). Class, Metaclass (types) aggregation of attributes, occurrences, values. Resource: Statement URI attributes / values.
* URI<Observable<Resource>> extends String;
* Resource<Observable<Statement>> extends URI;
* Statement<Observable<Kind>> extends Resource;
* Kind<Observable<Flow>> extends Statement
* Flow<Observable<Dimension>> extends Kind;
* Dimension<Observable<Measure>> extends Flow;
* Measure<Observable<Metamodel>> extends Dimension;
* Metamodel<Observable<URI>> extends Measure;
* Hierarchy: instance: previous statement, class: next statement, metaclass: occurrences. Class hierarchy: every class reified as instance of previous statement class.
* Activation / aggregation on layers.
* DCI Layers: dimensional, data, schema, behavior. CSPO roles.
* CSPO members: (C: Class, S: OccurrenceSubClass, P: SuperClass, O: SuperClass metaclass);
* Bound functions: aggregate / de-aggregate, functional activation, transforms, filters, predicates, kinds, typed (roles: dimensional data, schema, behavior / instance data / kind relations) traversal / joins, others.
* Versioning / CRUD: context relations determines current / role entities occurrences.
* Spring / OSGi Blueprints
* ServiceMix / Fuse
* RxJava / JAX-RS
* Declarative settings: nodes, routes, patterns / transforms functional flows (monads): from Metamodel schema / instances data (URIs, Resources, etc.). Metamodel hierarchy layers (endpoints / routes / messages: URIs / Resources) as common Blueprint / Spring components (schema monads classes context / roles, interactions instance data).
* Messages: encoding / addressing. Signatures / patterns: dynamic discovery / flows. Bound functions: schema / domain DCI: declarative templates encoded in Metamodel (transforms).
* Graph encoding (Metamodel quads): Context, SPO, Kinds. Sets (recursive statement, kind, SPOs).
* Metamodel addressing: inputs / outputs of normalized URIs dataflow flowing through hierarchy from URIs till Metamodel till URIs (context driven): Aggregation, Alignments, Activation.
* Backends handles Metamodel URIs IO via Metamodel metadata / functional contexts.
* Type promotion. Role promotion (behavior, interface). Deterministic materialization: loops (reifications / occurrences roles). Definitions (templates). Reifications. Quads, triples, pairs (metaclass, class, instance, states: flows, behaviors).
* Activation wizards: assets (resolution, forms, flows). Dynamic assets 'prompts'. Dialog 'protocols' (slots three alignment kinds in / out nested contexts). Assets: ActiveRecord / queues. Chrome / XUL / SWT / ZK runtimes. Jupyter notebooks.
* Layers (metamodel): uniform modelling of domains and applications / services (parser). DCI / MVC 'gestures' (IO / activation / user, service interaction) as activation messages. Encoding, resource augmentation, lambda wrappers (bound functions in descriptions from metadata). Uniform layer interaction models. Events / queues (server push).
* Microkernel: workflow / scheduler. Input / output plugins, kernel plugins. Declarative blueprints / layers (CDI / IoC / domains discovery).
* CQRS. Event sourcing (rehydrate entity, descriptions somewhere).
* Microservices: from aggregated domains descriptions (augmentation, declarative template extensions, declarative plugins).
* NodeJS: meta-web description framework. Activation (descriptors pointers / lambda commands, runat contexts layers integrations). Serverless (codats).
* Streams. Reactive Programming. Dataflow. Endpoints (behavior) / content (schema) metamodel node / layer modelling. Signatures. Frameworks: messaging, networking, events, routing. Metamodel Activation Protocol. APIs protocols. Implementations. Deployment: containers / custom blueprints, management APIs.
* Encoding: primes / primitives / metaclass, class, instance characteristics. Numeric systems / aggregation, groupings, comparable rels. Contexts: quads / tensors (recurs.). Embeddings. State (events / attributes /flows): aggregate kinds / processes / actions / dimensional transitions (data / schema / behavior types / templates).
* Framework: Dimensional / hypermedia application semantic modelling (HyTime / Topic Maps / SW). Abstract Metamodel layers (dimensional data, schema and behavior: templates). Rendering (services, apps: XUL / ZUL, SVG). XML XSL / XPath / XLink / XPointer / XQuery metadata. Plug backend layers (declarative templates / microservices).
* Sets, groups, categories, graphs (formalization / DM, RM).
* Schema, microformats (annotations / metadata), JSON-LD. REST HATEOAS / HAL from Metamodels (resources, rel / declarative discovery links, self links: lambda / runat messages semantics). Declarative descriptions / locators (discovery / signatures) frameworks.
* HyTime (dimensional addressing, traversal: gestures / events), TopicMaps, XML, XSLT, XPath, XLink, XPointer, XQuery. Discovery / signatures (arcs / nodes) / query data / node paths (XPath encodings). External metadata linkbase arcs (pointers) actuate traversals, referrer headers, HAL / HATEOAS 'self' rels. Graph aggregations (XLink / metamodels).
* App (proof of concept): expenses, organizer (soccer / medical appointments / generic tasks assistant / wizard, objectives). Shopping use cases: actors (customer / seller), supply chain. Campaigns (selling / buying objectives / flows).
* Formalization: metamodels / alignments (dimensional / layers) data / reference models (parsing). Canonical forms (graphs iso / rels). Quads (self occurs loops / recursive). Merge / alignments (id, attrs, roles: encode rels):  kinds / attrs / values. ML (graphs).
* Serialization / externalization. Serving models. Data URL scheme.
* Deploy: lambda, runat, serverless.
* Monads. Parsers. Field access interfaces.
* ANN / Deep pipelines / flows. Reactive streams.
* Embeddings (words). NLP.
* Statements 'ordering' (streams / events sourcing: declarative entities descriptions). Dimensions. Tasks / provenance: steps, states (behavior rels / templates).
* Big Data, SemWeb, ML, Graphs.
* ID Alignment: metaclass / class / instance kinds (attributes / values), occurrences graphs analysis from primitives to objects lattices (rels / roles).
* Metamodel (labeled arcs: occurrences).
* Bus address encodes input / output 'signatures' (registering / discovering functional 'domains': backends, services, predictions / activations). Location (expressions) resolvers.
* Bus address encoding: dimensional data, schema, behavior inputs / outputs (message) signatures. Locators: metamodel addressing expressions / parameters. Invocations: Locator message instances (knowledge domains functional hypermedia / browseable messages APIs).
* Data: price, information: price var, knowledge: price tendency. Dimensional data, schema, behavior. Layers.
* Stanbol, Any23, Metamodel, Teiid. OData. HAL. GraphQL. TMDM / TMRM / TMCL (Constraints / schema as data: Validating RDF Data, metamodel data / reference models).
* Encoding: Translate graph into dot notation keys / values property chains. Arrays / templates / variables / placeholders.. Indexing / search / alignments.
* Property graph encoding implementation. Metamodel, layers, persistence (index, registry, naming) search / algorithms. Instance, class, metaclass. Alignments.
* key1 = context.subject.predicate.object.attribute: '1234';
* key2 = key1.context.attribute: 'abc';
* Paths: [index, occurrence, context] expressions / resolutions.
* Schema: annotation kinds:
* Entity / ID.
* Class ID (ID Type: genre : ID, ID Value: male : Entity).
* Instance ID (ID Type: SSN : Entity, ID Value: 1234 : ID)
* Attrs / Rels (Class ID, Instance ID) pair.
* Contexts / Roles (Attrs, Rels pairs: Class ID, Instance ID context / role). Parent / child hierarchies.
* Schema: annotation(annotation\_id, annotation\_kind, annotation\_subject);
* Schema: class\_id\_type(class\_id\_type\_id, name); parent\_class\_id\_value\_id: two tables.
* Schema: class\_id\_value(class\_id\_value\_id, class\_id\_type\_id, value, annotation\_id, metadata);
* Schema: instance\_id\_type(instance\_id\_type\_id, class\_id\_value\_id, name);
* Schema: instance\_id\_value(instance\_id\_value\_id, instance\_id\_type\_id, value, annotation\_id, metadata);
* Schema: annotation\_subject(subject\_id, type\_id, value\_id, name); class / instance browse / refinements.
* Schema: metadata, class / instance types values JSON forms.
* Infer function / models from inputs / outputs.
* Domains. ERP / HMS / CRM / SCM / BI. Integrations (Activation).
* VM (RDF backend event driven, graph DB, TensorFlow shapes).
* Abstractions: Graphs, Edges / Statements, Nodes / Resources. Aggregations (Edge / Statement), Activations (Node / Resource). Edge / Node "expressions" (signatures) : declare aggregations / activations (Forms / Templates: possible Statement / Resource instance classes). Aggregation / Activations: Statement / Resource “occurrences”.
* Dataflow (declared routes / signatures: matching shapes / grammars) aggregations / activations.
* Aggregation: Statement(Resource) : Statement. Aggregates over Resource into Statement. Declare aggregation.
* Activation: Statement(Statement) : Statement. Declare activation.
* ID Alignment: Resource(Resource) : Resource.
* Attr Alignment: Resource(Statement) : Statement.
* Role Alignment: Resource(Statement) : Resource.
* Operations as Dataflow streams possible aggregations / activations. Translation / orderings / comparisons / aggregations.
* Models / graphs layers from Resource until Behavior, binded through semiotic Resource / Statement / Role IDs metagraph.
* Data (resources: syntax).
* Schema (dimensional: semantics).
* Behavior (application: pragmatics).
* Semiotic Resource / Statement / Role IDs metagraph.
* Naming, registry, index services APIs.
* Translation (domains). Subject, attribute, value, occurrence, role bindings.
* Primitive operations: sort in context, compare, aggregate, activate, alignments (ID, Attrs, Roles).
* Functionality: translate / integrate. Expose uniform distributed systems gateway. IO / facades (data, schema, behavior activation flows). Pluggable architecture (sources / endpoints: same aggregation, activation, alignment interfaces).
* Resource / Statement: Metamodel
* Template / Form: Declaration
* Activation / Aggregation : Behavior
* Data Metamodel
* Dimensional Metamodel
* Flow Metamodel
* Resource IDs Graph: Dataflow (Aggregations, Activations, Alignments).
* Executable Metamodels. Domains Translation. Unified Integration Dataflow Event Bus.
* VM / IO Connectors (client / service abstractions). Declarative Functional Data / Dimensional / Flow Metamodel Layers specifications (pub / sub streams) Templates / Forms. Activation / Aggregation "instances".
* Element: Observer / Observable (streams / filter / map / reduce).
* Node / Edge, Resource / Statement, Template / Form, Activation / Aggregation.
* Dataflow: matching element "templates" (grammars / signatures).
* Events: Element wise listeners (instance, class, occurrence). Produces new Elements (events).
* Events example: Forms produce Statements from Aggregations, Statements Aggregations and Aggregation Forms.
* Elements: publish / subscribe via addressing (observer / template signatures dataflow) to / from edges / nodes (Templates / Forms, Statements / Resources, Aggregations / Activations).
* Addressing: From Resource Quads Encoding. Resolve routes from / to via template signatures dataflow.
* Backends (support implementation of edge / node element level events):
* RDF Backend.
* Graph DB Backend.
* Reactive (Vert.x) Backend.
* TensorFlow Backend.
* Element API: Wrapper over backend implementation. Event handling. Base Metamodel hierarchy (grammars / schema: Dimensional / Role Kinds, data: layers, behavior / flows: Resource Graph IDs).
* Order (sort in context), comparison (class / instance ID): "common denominator" (jump / iterations). Activation /Aggregation. Read / Write Attributes.
* Data (Nodes / Semiotic), Information (Templates / Dimensional), Knowledge (Resources / Layers).
* Processor : Observer, Observable.
* Exchange : Subscription. Processor.
* Processors (Alignments):
* Nodes, Edges, Routes (hier.).
* Templates, Forms, Aggregations (hier.).
* Resource, Statement, Activations (hier.).
* Processors: instance of (type object) hierarchy. Activation, role kinds: Aggregation. Aggregation, paths: Routes.
* Declarative Aggregations: hierarchy type object pattern.
* Models: Processors. Collections (observables) of Activations, Aggregations, Routes.
* Input URL Quads: Resource(s) in Statement: Activations (Kinds). Classification alignment.
* Templates (same Resource) in Forms (same attributes / values) : Aggregations (Statements, from Activation Kinds). Infer, link information models. Regression (role in context) alignment.
* Nodes in Edges : Routes (from Aggregations). Resource IDs (Semiotic) Metamodel (models graph / occurrences) : Routes (recurse Activations, feed Statements for activatable Resource layers). Infer, link data models. Clustering (link / attributes alignment).
* Models:
* Activation: (Activation, Statement, Resource, RoleKind);
* Aggregation: (Aggregation, Activation, Form, Template); Form, Template: statements, kinds / resources of Activation RoleKind occurrences.
* Routes: (Route, Aggregation, Edge: Activation, Node: Form);
* Models aggregation metadata (describes models / models values):
* Activation Kinds (schema / dims classification : metamodel layers): (Kind / Resource : unit, super Kind : dimension, Activation : measure, Kind : value).
* Aggregation layers (context / role regression): (Template : player, Template : occurring layer, Aggregation, Template : occurrence value).
* Paths attributes (attributes augment / clustering): (Node : player, Node : occurrence / context, Route : attribute, Node : value).
* Template / Form (Kinds) upper root hierarchy: Entity, Kind, Class, Flow, Behavior.
* Message wrapper (functional / monads / elements events callbacks / type system / streams / signatures / headers). Exchange / Protocol.
* Model declarative Aggregations:
* Entity, Kind, Class, Flow, Behavior : Facts.
* Inferred models:
* Dimension, Unit, Measure, Value : Schema.
* Context, Concept, Sign, Object : DCI / App.
* Context, Occurrence, Attribute, Value : Grammar
* Alignments:
* Classification (class / ID).
* Regression (role in context)
* Clustering (links / attributes).
* Models: trained / augmented on each Message exchange (online training). Events: Processor IO implementation.
* Models: Collections (observables) of Activations, Aggregation, Routes. Perform inferences. Parameterized Messages.
* Map Resource, Statement / Template / Forms, Node / Edges to objects / tensors (events).
* Reactive API (Message streams : Protocol) dataflow / IO:
* Activation (in quads) - Aggregation (eg.: Entity / Class) - Routes (augment attributes) - Aggregation (e.g.: Class / Entity) - Activation (out quads).
* Protocol:
* Upwards: Statement::toLayer(model);
* Downwards: Statement::fromLayer(model);
* Message pattern builder (Activation). Message as (observable) resource “persists” as an endpoint stream till yield results (services implementation).
* Message Exchange functional transforms:
* Order
* Comparison
* Iteration
* Jumps
* Backend connectors (Resource / Statement Layers / IO : Exchanges):
* RDF / Graph DB (event driven) Backend connector.
* Reactive (Vert.x) Backend connector.
* Services: Spring / OSGi (events) Backend connector.
* TensorFlow (events / dataflow) Backend connector.
* RDBMS, Messaging, Services Backend connectors.
* Feature columns: FCA lattice bitstring.
* Deployment:
* Ont DIDs: Encoding. Addressing, Routing. Distributed logs / routing layer (Alignments : Transactions / Exchanges Messages logs). Endpoint client bindings Processor (resolve service node address / message pub / sub: streams / events logs). Business (Metamodel) layer message abstractions (IO: import / export converters, DCI / DOM / object mappers / content types handlers).
* Services layer. Nodes Backend. Message routes / protocol resolution (based on DIDs addresses / headers / logs).
* Nodes Runtime. Business (Metamodel objects) layer. Data, information, knowledge models (aggregated). Resource / Statement : Template / Form. Factory / Builder (for each model): Forms (of layers) from URLs / objects factories. Layer facade: Aggregate models / perform alignments / de-aggregate models (in / out Message Dataflow).
* Dataflow: Models (layers) observing each other, Resources observing occurrence Statements, Statements observing occurring Resources. Forms CSPOs specialization (pre / post aggregation / alignments).
* Forms CSPOs specialization: retrieve / consolidate / determine equivalent CSPOs, Kinds between Messages. Resolve common entities in context. Aggregation / inference facade.
* Activation: determine kinds / resources from equivalent IDs. Ontology matching. Classification.
* Aggregation: determine declared aggregation layers occurrence.  Determine declared Aggregations (Entity, Kind, Class, Flow, Behavior). Dims. Schema. Clustering.
* Paths:  determine role in context of occurrence in path. Regression.
* Retrieve requested Form hierarchy from performed alignments (layers). Build response graph.
* Apache Jena / [rdflib.js](http://rdflib.js): RDF Store Backend. Model persistence layer. Persists Business layer inputs / inference layer outputs Metamodels.
* TensorFlow / Aggregations / Alignments inference: Nodes Engine. Model inference layer.
* Layers: async event driven interfaces (Exchange contexts: Messages). IO flows Processors.
* Activation (Kinds) / Aggregation (Upper) / Alignments. In / out IO. ResourceManager feeds Resource / Statement. Events handlers dataflow. Upper ‘kinds’: roles.
* ResourceManager : Observer, Observable. CRUD source / dest notifications / sync. Lazy fetch (query). Manager : Resource having occurrences / occurring in Statements.
* Main / upper hierarchy:
* Kind : Entity : Statement : Resource.
* Behavior : Flow : Class : Kind.
* Resource events:
* onContextOccurrence
* onSubjectOccurrence
* onAttributeOccurrence
* onValueOccurrence
* Statement events:
* onEntityOccurrence
* Entity events:
* onKindOccurrence
* Kind events:
* onClassOccurrence
* Class events:
* onFlowOccurrence
* Flow events:
* onBehaviorOccurrence
* Bound functions (getters / setters CSPO / Kinds):
* Class / Interface API (config deployment and application instance graphs declaratively in model statements / hierarchy): Peer / Node, Model, Graph, etc. (Resources). Observe ResourceManager, augments on new Resource, notifies ResourceManager on event updates with augmented data. Dataflow bindings on configuration statements.
* RESTFul API Resource endpoints for native RDF application configuration (backends). Events handlers. Example: RDBMSResourceManager.
* Resource (upper hierarchy) : URL.
* Resource<URL>. Events: HTTP verbs (Processor) bound functions. Navigate representations (body / content type "activation" bound functions) / verbs. Headers. Ont DIDs, endpoints (uniform API)a. Resource (onOccurrences) : bound functions.
* Resource.contexts / onContext / getContext (fMap): Stream<Statement>;
* Resource.subjects / onSubject / getSubject (fMap): Stream<Statement>;
* Resource.attributes / onAttribute / getAttribute (fMap) : Stream<Statement>;
* Resource.values / onValue / getValue (fMap) : Stream<Statement>;
* Resource.kind(Statement) : Kind;
* Statement.kind(Resource) : Kind;
* Resource.occurrences(Resource) : Statement (hier parent / child; example: Entity in Kind).
* Resource.occurring(Resource) : Statement (hier child / parent; example: Kind Entity).
* Upper ontology aggregation: monadic wrapper for lower level contexts. Example: Behavior over Flow augmentation. Resource.fMap(Resource) and events / streams logic.
* Promise (async) based pipelines for layers augmentations.
* Augmentation: Activation, Aggregation, Alignment. According Resource URI content type mappings. Registered processors. (providers extract Statement from representations). Example: Image / Face (attributes / links, coords qualified); Face / Gender (classification); Face / Subject (ID); Subject / Occurrence (context / role).
* Content types: image/png;face, text/xml;faceImgCoords, text/rdf;faceImgPoints
* Nodes: Java / Vert.x / NodeJS.
* Command line (container / boot) nodes: Java. Ont DIDs DLs (rendezvous nodes). Plug backends. Clients. Augmentation / Resource endpoints resolution (local / remote). Implement event bus protocol (Message handling).
* JS nodes: NodeJS microservices (container / plug backends / clients). Augmentation / Resource endpoints resolution (local / remote). Implement event bus protocol (Message handling).
* Nodes: Browser. Clients. Client container plug "edge" / app domain backends. Augmentation / Resource endpoints resolution (local / remote). Implement event bus protocol (Message handling).
* Nodes: virtual distributed address space. Local / remote transparency. DIDs service. Endpoint location translation. Resources events dispatch. Preferable Node by capabilities / location. Async DIDs DL updates. Node idempotence: every Node may perform Augmentation (update Resource by DID service). Event bus (Resource URIs).
* Dimensional: comparisons / distance. Nested previous / current / next measures (next hour, next minute). Contexts: next distance at next time at current speed, next day at day of week, etc.
* Serialize: Distance, Previous, Next (Value)
* (Value, Distance, Previous, Next);
* (Measure, Value, Distance, Previous);
* (Unit, Measure, Value, Distance);
* (Dimension, Unit, Measure, Value);
* (Concept, Dimension, Unit, Measure);
* Encode layered statements orderings: dimensional / upper ontologies. Kinds / Roles. Flows / Templates / Forms. Previous, current, next (octal value temporal order feature?).
* Primitive values: self, the, this, that, inside (here / that), outside (here / that), after, before, during, etc. Value variables / placeholders / wildcards.
* Assert class hierarchies, containment relation, order relation, roles relation (occurrences / metaclass).
* Unify (link) upper / dimensional ontologies.
* Upper: (Behavior, Flow, Class, Kind); Entity
* Dimensional: (Concept, Dimension, Unit, Measure); Value : Entity
* Type, State.
* Event : Measure, Kind.
* State: Concept, Behavior.
* Axis: Dimension, Flow.
* Type: Unit, Class.
* Event: Measure, Kind.
* Grammar / Type System:
* (Event, Event, Event, Event);
* (Type, Event, Event, Event);
* (Axis, Type, Event, Event);
* (State, Axis, Type, Event);
* (Object, State, Axis, Type);
* Primitives: disabled: true / enabled: false. Complements / supplements in contexts.
* Sets, categories, functional models (reference / data: upper ontologies common abstractions). Formalization.
* Messages (Events): Declarative transforms specifications (capabilities) / instances (exchanges). Activation, Aggregation, Alignment declarations.
* Message<Resource<? : URI>, R : URI> : Applicable;
* Visitor: Message::apply(Resource<?>) : Message<Resource<R>>; Reactive Dataflow.
* Upper ontologies classes instantiated as Resources (Concept, Behavior, Object hierarchies : reify / reflection).
* Metamodel ontologies
* Contexts / Interactions:
* (Entity, Statement, Resource, Resource);
* (Kind, Entity, Statement, Resource);
* (Class, Kind, Entity, Statement);
* (Flow, Class, Kind, Entity);
* (Behavior, Flow, Class, Kind);
* Dimensional:
* (Value, Distance, Previous, Next);
* (Measure, Value, Distance, Previous);
* (Unit, Measure, Value, Distance);
* (Dimension, Unit, Measure, Value);
* (Concept, Dimension, Unit, Measure);
* Grammar / Type System:
* (Event, Event, Event, Event);
* (Type, Event, Event, Event);
* (Axis, Type, Event, Event);
* (State, Axis, Type, Event);
* (Object, State, Axis, Type);
* State: Concept, Behavior.
* Axis: Dimension, Flow.
* Type: Unit, Class.
* Event: Measure, Kind.
* Formalization (class / metaclass / instance):
* (Context, Occurrence, Attribute, Value);
* (Context, Sign, Concept, Object);
* Sets, categories, models (reference / data).
* Transforms: translate source resources (observable URIs) into metamodel ontologies. Activation, Aggregation, Alignment.
* Transform, Message / DB Resource:
* (Table, PK, Column, Value);
* Message<<Resource<URI : DBURI>>, R : Statement>
* Applicable: CRUD / Query Messages.
* Transform, Message / Hierarchical (JSON / XML Resource):
* (Context, Parent, Value, Children);
* Message<<Resource<URI : EndpointURI>>, R : Statement>
* Applicable: HAL / REST Messages.
* Transforms: translate source resources (URIs) into metamodel ontologies. Augmentation: Activation, Aggregation, Alignment.
* Activation, Aggregation, Alignment : Message specs (reified ontologies / metamodel / augmentation resources reified "verbs"). Reactive dataflow.
* Message<<Resource<URI>>, R : Statement>; R: Activation, Aggregation, Alignment (Augmentations).
* Messages dataflow order: inputs, outputs (static / runtime) type flows. Comparison (dispatch) by specialization hierarchies, super: first, sub: next (order in contexts traversal : execution trees / graphs “definitions”, metamodel “activated”. Runtime content type flows.
* Quad Statement Equation Encoding:
* aX⁴ + bY³ + cZ² = dW;
* a, b, c, d: classes (SPOC).
* W, X, Y, Z: instances.
* Powers: statement SPOC roles.
* Terms: occurrences (values).
* X (object) is Y (predicate) for Z (subject) in W (context).
* Operate (filter / sort / map) over features. Satisfy context dW (calculated desired results class / instances) SPOs. Messages / dataflow: aggregate possible mapping / filter verbs / results into reified resources. Alignments.
* Identifiers: embedded features (substituted coefficients and variables).
* Normalize Terms (SPOs) class / instance according context (dW) class / instance. Normalize dW Term.
* Alignments: Class / ID, Attributes, Roles (order, containment, other relations) verb Messages. Message of possible context input / desired output contexts.
* Nested contexts: ontologies hierarchies.
* Nested contexts: ontologies hierarchies. Message aggregation: composite contexts: Message verbs, predicates, subjects, objects composition (Message as template for context resolutions). Term class equals Term variable: class Term.
* Contexts in model hierarchies (behavior, dimensional, grammar): aggregate / map class / instance values on Activation / Aggregation (map URIs / IDs: DIDs, endpoints).
* Encoding: quad statement.
* Class Terms: statement templates.
* Resolution via execution graph dataflow. Alignments (encoding as verbs: resources / messages):
* Class / ID: Common upper / same (equivalent) contexts results.
* Attributes: Terms applying. Predicates satisfying quad encodings results.
* Roles: Role in contexts. Context as object Term in predicate role for subject context satisfying enclosing context statement results.
* Comparison: Sort alignments execution graph dataflow according context / verb resources / messages order. Iterations. Jumps. Comparison contexts (alignments): Class / ID, Roles, Attributes. Match next contexts. Nested contexts.
* Augmentation: Activation, Aggregation, Alignment declarative statements (resources, messages / verbs: resource application, reified metamodels). Statements, Terms, Class Terms. Variables, placeholders.
* Message<T, R>;
* T, R: Resource<URI>;
* Apply resources: T : Person (class), R : Employee (kind) : Employees (kinds, occurrences).
* Apply resources: T : Employee (kind), R : Person (class) : Person occurrences as Employee (classes, occurrences).
* Metamodel graphs. Dynamic activation, aggregation, alignment (execution graph dataflow) resource declarations.
* Execution (dataflow) graph: Messages (transforms) resource declarations (activations, aggregations, alignments). Member (attributes) access. Graph statements describe graphs available interactions (messages / transforms). Interactions invoked according ontology alignment rules / metamodel aggregation (order / contexts). Entailed results appended (activated, aggregated, aligned) into originating models.
* Ontology alignment rule: (Occurrence, Context, Class, Metaclass); Mappings (message instances / transforms) for relations between metaclass / class / instance.
* Member access: Mappings between Context : T and Context : R (URI member specialized) resources.
* Augmentation: Activation, Aggregation, Alignment described as dataflow execution message flows (reify message resources in metamodels).
* Execution graph:
* (Message, T : LHS, R : RHS, MappingValue : R);
* (Interaction, Message, T : LHS, R : RHS);
* (Context, Interaction, Message, T : LHS);
* (Dataflow, Context, Interaction, Message);
* Flow ordering by Message inputs / outputs. Activate, Aggregate, Align: learn dataflows.
* Reference model: model abstractions used to describe models defined itself in terms of (a) model(s).
* Mapping to / from metamodels / model instances representations. Ontology, Inference, Alignment, Constraints, Shapes. Encodings.
* Subjects / Contexts: Occurrences / Roles.
* Occurrences / Subjects: Contexts / Roles.
* Subjects / Roles: Contexts / Occurrences.
* Attributes / Values: from entities types applicable for each term (example: occurrence attrs / vals in corresponding ctxs).
* Metaclass, Class, Instance.
* Data: items / attributes / values. Classification, types (activation).
* Information: Data items in context. Roles. Items relations (aggregation).
* Knowledge: Information evolution. Predictions (alignments).
* Topics:
* Math.
* Python.
* Model: ANNs. TF. ML. Dataflow graphs (Resource tensors). Translation / Alignments. Semiotic Graphs (data, schema, behavior).
* Big Data. Data Mining. Spark. Solr. Lucene. Index, naming, registry.
* SQL.
* Parsing. Grammars (logic, attribute, etc.).
* LISP. Data structures (graphs).
* Theory: information (retrieval), sets, categories, groups.
* Application: Reactive / Spring / OSGi / RxJava / Vert.x / Dataflow (Resource IO / Activation) / Event driven. Addressing / resolution. Render model layer knowledge semantics into message protocols).
* Scala (signal processing: case classes). Functional Java (generics, etc).
* Business (REST): HAL / HATEOAS. GraphQL. Render application layer messaging semantics into endpoint contracts).
* Frontend: HTML5 / CSS / SVG (XUL / ZUL). Others (render business layer contexts / interactions into gestures).
* NodeJS. OO / Functional JavaScript.
* LIEC (emulation). C / C++.
* 1) XML: XSL / XPath / XLink / XPointer / XQuery.
* 2) Semantic techs. ISO (upper, data / reference models). KR. Logic. Rules. Inference. Graphs (dataflow). Tagging (annotations metadata): Class / ID, Links / attrs, Context / roles of subjects / occurrences (Content type activation / annotation: document describing picture occurrence with annotations in context, document / picture role: diagnose / evidence, prescription / documentation, invoice / originating sales order, profile picture / holidays picture. Roles in contexts activation "flows": orders, invoices, etc.). ERP upper onto. Dataflow (operations: flow transitions / links, addressing according states).
* Linked and addressed "activated" content (subjects / occurrences). Profile: profile picture, contacts, messages; Holidays: holiday's picture, place, itinerary. Identify same picture in both contexts. Infer rels.
* 1) & 2) plus addressing / resolution:
* Translation / Activation examples: link pictures with DB records (in context / roles / occurrence, attributes of subjects). Identity, Attributes, Roles Alignments.
* Expose, consume (endpoint / service, datasource / backend): same interfaces (dual behavior). Observer. Service resource request / response to events, datasource reacts / generates events. Functional dataflow / models (activation bindings).
* Grammars: syntax / semantics / pragmatics. Statements: attribute / logic grammars. Logic, rules embedded.
* Semiotic (syntax / grammar: data, semantic: dimensional, pragmatic: behavior) inferences (activation / grammars). Models (data / metamodel, schema / dimensional, behavior / application) productions / rules (Resource IDs Graph / Activation statements). Parsing (dataflow graph states).
* Dimensional: encode relations, conditionals, flows (containment, space, temporal, causal, states, part, whole, roles, etc.) via orderings and ID Graph statements.
* Each node has internal networks / graphs and exposes its graphs / models through DIDs. Networks nodes uses DIDs internally to retrieve Resource features. Services. (TensorFlow / Vert.x / Ont DIDs).
* Dataflow: Resources / Activations graphs / networks. Declarative IDs graph over data / schema / behavior graph models. Activation: apply Resource in role (operation) in statement (context) according IDs graph dataflow activation flows (production rules). CAM / declarative transforms specifications. Order relations encoding (models / IDs graph).
* Domains. Models (data : Metamodel / schema : Dimensional / behavior : Application). Translation. IDs Graphs.
* Order relation encoding: previous / current / next statements (primitives comparator ternary operation: expected next, self, the, this, that values assigned to roles in contexts / features graphs). Cycles (digit positions, nested comparators).
* Order: Layers Statement / Contexts comparison (Dimensional). Superclass: previous. Subclass: next.
* Octal encoding: comparison result (CSPO Statement).
* Graph: encode computation / control structures. Apply transforms (ops) to ordered / filtered streams. Control flow: next declarative resource / operation dataflow (transform results activation).
* Turing complete: states (initial / halt). Read: eval state / input, action (write, jump). Conditional branching / iterations.
* CAM: content / adresses, data. Index. Ternary X: don't care. ALU (PC). Comparator.
* Monads. Lang parser.
* Activation: Map / Reduce: graphs / activation. Filter / sort (functional map apply op over a functor : Template). Reduce / summarize / aggregate according hierarchy traversal combination operation (fold hierarchically CSPOs : Form).
* Map / Reduce: activatable behaviors. Dataflow Activation nodes.
* Control dataflow: activatable Activations.
* Activation: Map (apply) Resource over Resource : Resource. Grammar possible production statements from common hierarchical roles between SPOs of operands. Reduce, fold possible statements by relevance Resource. Possible activations: possible grammar productions given contextualized order of statements in context (dataflow). Being able to extract new knowledge statements from existing ones (Aggregations, ID, Links / Attrs, Ctx Roles Alignments). DIDs transactions: event sourcing features.
* Distributed (aggregation / alignments):
* Map(k1,v1) → list(k2,v2)
* Reduce(k2, list(v2)) → list(v3)
* Resource IDs Graph (metagraph for models bindings: data / schema / behavior Aggregation, Activation, Alignment):
* Resource (cls:id), Statement, Role.
* Statement (cls:id), Resource, Role.
* Role (cls:id), Statement, Resource.
* Models (layered):
* (Context, Occurrence, Attribute, Value); Metamodel. Data.
* (Dimension, Unit, Measure, Value); Dimensional Graph. Assert values for every sub-measure (5min: 4min, 3min, etc.). Express equivalence (60min, 1h). Express context (tomorrow, friday). Schema.
* (Context, Concept, Sign, Object); Application Graph. Behavior.
* Resource occurrences: mode (tense / person / primitives: the, this, that, self, etc) "declination"
* Combinatorial generalization in graph networks.
* Signal processing. Case classes.
* Specification (CSPO + APIs) declarative components (map / reduce / fold event handlers / callbacks).
* Comparison operators (map / filter / fold) : Specification API (Dimensional / Graph contexts).
* Template: Resource Specification.
* Form: Transform Specification.
* Activation (Template: input / Form: binding / Template: output). Capability (interaction) definition. Capability: Activation(s) instances (interactions). Available Message exchanges. Exchange: Message (data) blueprint.
* Activation: query, filter, transform (inference / aggregation / alignments).
* Map / Reduce: activatable behaviors. Dataflow Activation nodes.
* Application Graph:
* Behavior of Resolver / Runtime, Specification / Component, Activation / Capability, Exchange
* Layers:
* (Resource, Occurrence, Attribute, Value);
* (Statement, Resource, Occurrence, Attribute);
* (Entity, Statement, Resource, Occurrence);
* (Kind, Entity, Statement, Resource);
* (Class, Kind, Entity, Statement);
* (Flow, Class, Kind, Entity);
* (Behavior, Flow, Class, Kind);
* Triples / Quads Tensor encoding shapes: triangles / squares (points in dimensional space, recursive from parent context shapes). Operations / activation functions: based on points / sides lengths / angles operations. Vertex position: seed / bias. Shapes compose graphs (arcs: sides / nodes: vertexs: points given each context / parent Tensor). SPO Vertexs. Context vertex (square inside triangle). Nodes containing sub-graphs (as contexts). Similarity: euclidean distance. Operations: order in contexts.
* Project:
* Environment:
* Frameworks / Libraries. Docs.
* Runtime. Tools. Services:
* Jersey / CDI. Spring.
* RDF4J (OpenRDF Sesame).
* R2RQ. Teiid. Metamodel. Any23.
* Predictions: ML / BigData (Spark).
* Index: Lucene / Solr. Quads VSM.
* Registry (Zippers): hierarchical key / value. Map Reduce.
* Naming: DIDs Services. [schema.org](http://schema.org), [sameAs.org](http://sameas.org). Matching (mappings). WordNet, OpenNLP, Wikipedia.
* Reactive (Bus): Vert.x
* Dataflow: SCDF Pipelines.
* Designer: OpenRefine. Protege Web. Editors / browsers (Forms / Flows).
* Protocols (SAILs): SPARQL, JSON-LD, JDBC (OGM / OData: Driver DatabaseMetadata). Messages (Functional / Augmentations) Declarative Services (templates).
* Clients:
* Debug Console.
* Declarative UI: ZK / XUL Templates (Content types / Components activation).
* Services APIs: Declaratively stated services: templates / queries.
* IDE. Runtime Deployment (Application).
* Clients: OData / JDBC / WS-\* (CXF) / HATEOAS / etc.
* Lectures. Topics.
* Schedule. Topics.
* Triple store backend. RDF4J:
* Quad Layers SAIL. Core layers schema. IO: RDFS / OWL (classes, types, sameAs, etc.). SPARQL. Upper Ontology.
* Objects Layers SAIL. IO: Objects (Layers / DOM / OGM) HATEOAS.
* Functional Layers SAIL. IO: Augmentations: Browse / Navigate Model (Objects / Monads). Message Driven.
* Augmentations / Domain Services / Dataflow SAIL. IO: Messages built in Functional SAIL / Augmentation Contexts results streams. Contexts: Model Contexts, Domains Services REST Contexts URIs (APIs), connector / clients plugins (signatures). DIDs Domain Services: internal URIs. DIDs / URI mappings / APIs. Signatures. Bus.
* Augmentations: Perform Aggregation, Activation, Alignment according input Message. Update Model Contexts (upwards), Occurrences (downwards) according Augmentation stream results (Contexts). Enqueue further dataflow messages.
* Dataflow (Predicates, Mappings, Functions) domain / range (signatures). SAIL registry (types / kinds bus). Bus: topics, queues. Reactive interfaces. Dispatcher.
* Services SAIL I/O: Connectors / Clients. Distributed nodes Connector sources.
* URI: Jersey / CDI APIs. Persistence interface template methods.
* Layers APIs.
* Statement URI: Layer URI + Layer URI instance ID.
* Layers CRUD tests.
* Functional APIs.
* Functional tests (Inside Flat Map: dataflow over Object APIs):
* Predicate tests. Activation.
* Mapping tests. Aggregation.
* Function tests. Alignment.
* (MessageDomain, Predicate, Mapping, Function);
* Flat Map: Dataflow chaining over Message transforms and dataflow results.
* Augmentations performed over individual Message roles in dataflow and in dataflow with other Message roles (APIs / contexts).
* Message: Functional wrapper of Layers Contexts hierarchies statements. Hierarchy: template methods, predicate, mapping, function behaviors (inside flat map and on message statement role positions: internal dataflows).
* Predicate, Mapping, Function, URI, Statement, Value. Root hierarchy types.
* (Statement, OldEmployees : Predicate URI / Activation, SalaryUpdate : Mapping URI / Aggregation, Percentage : Function URI / Alignment
* Monad::of(URI instance : hier);
* Monad::flatMap(Statement) : Monad (of Statement hier).
* Dispatch: each layer instance consumes matching or forwards upwards (layers signatures / zippers) incoming messages. Resulting Message enqueued for further processing.
* URI::onMessage (template methods: context, occurrence, attribute, value, role, contexts, occurrences, etc.).