* Reactive Activation: Inferences. Model Rules: N3 / Turtle / DSL. Templates: Resources, Kinds, Contexts Encoding (roles) for Functional Reasoning (Predicates: schema / values)
* XML / XSLT DTDs / XSD. RDFS / OWL OGM (Templates Encoding).
* Alignment Inferences (Functional Predicates): sameAs, greaterThan, lessThan, equals, partOf, parentOf, siblingOf, previousOf, nextOf, roles (schema / values).
* Functional Composite inferred Predicates:
* greaterThan([a.age](http://a.age), [b.age](http://b.age)) : older(a, b) : Activation. Matching, Sort.
* Predicates: Templates. Resources, Kinds, Contexts Encoding (Function predicates argument mappings). Composite from primitives / roles (Contexts).
* [business.products.premium](http://business.products.premium)
* Inference (Functions same results) Ontology Matching.
* AI for Understanding Human Goals
* "In the quest to capture ... social intelligence in machines, researchers from MIT’s Computer Science and Artificial Intelligence Laboratory (CSAIL) and the Department of Brain and Cognitive Sciences created an algorithm capable of inferring goals and plans, even when those plans might fail."
* "... ability to account for mistakes could be crucial for building machines that robustly infer and act in our interests ... Otherwise, AI systems might wrongly infer that, since we failed to achieve our higher-order goals, those goals weren’t desired after all. We’ve seen what happens when algorithms feed on our reflexive and unplanned usage of social media, leading us down paths of dependency and polarization. Ideally, the algorithms of the future will recognize our mistakes, bad habits, and irrationalities and help us avoid, rather than reinforce, them."
* <https://scitechdaily.com/new-mit-social-intelligence-algorithm-helps-build-machines-that-bette>
* ("Inference" is used broadly herein to mean any rule or procedure that produces new assertions from existing assertions -- not just conventional inference engines or rules languages.)
* Furthermore, applications often need to perform custom "inferences" (or data transformations) that are not convenient to express in available (non-standard) rules languages, such as RDF data transformations that are needed when merging data from independently developed sources having different data models and vocabularies.  And merging independently developed data is the \*most\* fundamental use case of the Semantic Web.
* One possibility for addressing this need might be to embed RDF in a full-fledged programming language, so that complex inference rules can be expressed using the full power and convenience of that programming language.  Another possibility might be to provide a convenient, standard way to bind custom inference rules to functions defined in a programming language. A third possibility might be to standardize a sufficiently powerful rules language.
* Here’s a JavaScript-based language for path queries, which reduce things such as “the user’s list of friends” to three words ([user.friends.label](http://user.friends.label)) instead of a SPARQL query:
* – <https://github.com/solid/query-ldflex>
* – <https://solid.github.io/ldflex-playground/>

**To do**

* CQRS. Monads Functions (domain / range) CUD Commands, R Retrievals applicable in contexts / roles: DDD (signatures / dataflow).
* CoSQL. Duals. LinQ / DSL / Parser Combinators. Templates.
* Aggregation:
* Parse Sets Model input Statements into AST Monads. Parsed Model Execution: Parse Event Bus I/O Model Message Monads Dataflow (sync: AST Events).
* Alignment:
* Encoding / addressing Sets Model population. Versioning. Parsed Model Execution: Parse Event Bus I/O Model Message Monads Dataflow (sync: AST Events updates).
* Activation:
* Alignment Augmented DOM Model: Parse Sets Aggregation AST : DOM AST Monads. Parsed Model Execution: Parse Event Bus I/O Model Message Monads Dataflow (sync: AST Events).
* Sets Model: (hierarchy Resource root, Kinds, Contexts children). Upper Ontology (DDD).
* Contexts: CSPO Layout (Kind Roles): Functions (Interactions).
* Statement: Data
* Mapping: Context
* Transform: Interaction
* Kinds: Roles (Contexts).
* Subject: Data
* Predicate: Context
* Object: Interaction
* Resources: Values (Data).
* Subject: Data
* Kind: Context
* Resource: Interaction
* Monadic Wrapper: Category.
* Wrapped: ASTs.
* Function: Domain / Range
* Event Bus: Message Monads / Functions.
* Core Models AST Functional Dataflow Traversal:
* CQRS Models Protocol: Sets Model Monads / Functions Events encoding / addressing:
* CUD: (Context, Statement, Mapping, Interaction);
* Context::Statement::Mapping::Interaction Aggregation. (CUD).
* (R), Selector: (Context, Kind, Statement, Resource);
* Context::Kind::Statement::Resource Aggregation. (R: Selector).
* Message Model Functional Dataflow Traversal:
* Encoding / Addressing Message Model:
* (URN, Statement, Kind, Resource); Subject Position: Query / Command.
* DOM Message Model:
* (Class : Instance, Instance, Attribute : Class, Value : Instance);
* Class::Instance::Attribute::Value.
* Models Integration: Core Upper Metamodel (Encoding / Addressing). Models. URNResource: Endpoints, Reactive HATEOAS Resource Monads (Categories), Functions declarations (domain / range signatures dataflow).
* Streams in Context. Functional Reactive HATEOAS Monads / Functions. Verbs / Types reified as Resources. Alignment Augmentation.
* Identity Transform (Context): Retrieval.
* Template = Context: Create.
* Update / Delete: Versioning.
* Models ASTs Monadic domains Functions.
* Functional API (Monads Functions / Wrappers (Domain / Range): Sets Object Model:
* Example: Resource<Subject>, Subject<Resource>;
* CoSQL. Duals. Meijer.
* API Functions (domain / range: individual subjects / streams in context):
* getResource / getResources
* getKind / getKinds
* getContext / getContexts (Statement, Mapping, Transform)
* getContext / getContexts (CSPO)
* getSubject / getSubjects
* getAttribute / getAttributes
* getValue / getValues
* getMetaclass / getMetaclasses
* getClass / getClasses
* getInstance / getInstances
* getContext / getContexts
* getRole / getRoles
* getOccurrence / getOccurrences
* Merge TOCs.
* Merge Contents.
* Content / Topics:
* Concepts, Design, Architecture.
* Add bibliography / tools use cases / components. Bookmarks, Lectures. Notes: Scrapbook.
* Concepts:
* Data, Information, Knowledge.
* Data, Schema, Behavior.
* Models: Layers Message IO Dataflow Bus.
* Models: Sets Contexts, Kinds, Resources Layer.
* Events Sourcing / Models Bus IO.
* Input Message Augmentations:
* Aggregation: Populate Sets.
* Alignment: Addressing / Encoding / Matching.
* Functional Activation Dataflow API:
* Resource Monad
* Kind Monad
* Context Monad
* Message Monad
* Event Monad
* API: Dataflow:
* API: Command. CQRS (CUD, R): Context (Mapping Contexts).
* API: Event: Command / Message (Context, Template: D, Mapping:  C, Transform: I);
* Dataflow: Event dispatching. Event Message / Command Context augmentation.
* Activation: Topics reacts to Events according API. Context, Kind, Resource Chain of Responsibility. Performs CUD/R and a response stream relevant to the operation performed.
* API: onEvent(Event) : Event. Order / Comparisons / Workflows.
* Activation: Statements.
* Template Data Roles (Kinds) selectors / predicates Matching Statements. Data.
* Activation: Mappings.
* Statements Matching Mappings. Schema / Context.
* Activation: Transforms.
* Mappings Matching Transforms. Behavior / Interaction.
* API: Core Model Bus Topics: Contexts, Kinds Resources.
* API: Core Model Transforms / Mappings Functions.
* API: Dynamic Model (instances) Bus Topics: Resources, Kinds, Contexts.
* API: Dynamic Model (instances) Transforms / Mappings Functions.
* Models: Addressing / Encoding / Matching Layer.
* ResourceURNs Occurrences (Subjects):
* (ResourceURN, Resource, Kind, Context);
* (ResourceURN, Resource, Context, Kind);
* (ResourceURN, Context, Kind, Resource);
* (ResourceURN, Context, Resource, Kind);
* (ResourceURN, Kind, Context, Resource);
* (ResourceURN, Kind, Resource, Context);
* Addressing: Model Traversal: MapReduce
* ResourceURNs Contexts, Resources, Occurrences IDs Addressing / Encoding:
* ResourceURN : (ContextResourceURN, SubjectResourceURN, OccurrenceResourceURN);
* Matching: ResourceComparator(s).
* Models: DOM OGM / DCI / DDD / CDI: Restful objects Layer.
* RDF DOM OGM / DCI / DDD / CDI: Sesame Elmo.
* Resources / Kinds / Contexts: DOM / DCI / DDD Subjects, Concepts, Mixins, Behaviors.
* Functional Dataflow API. Model / Events Bus sync.
* DCI: Qi4j / Apache Zest (RDF / KeyValue / EAV EntityProvider).
* Dynamic Functional Contexts: Scala DSL Message Dataflows. Monadic Parser Combinators.
* Behavior / Factory: Roles. Parameterize (domain context) monadic functions applications.
* Augmentation: Layers (occurrences, aggregations). Quads.
* Upper Ontology Roles Aligned Object Models. Templates.
* Roles: Metaclass, Class, Instance: Resource Occurrence, Resource, Kind, Context.
* Meta Model:
* Object Model:
* Labeled Property Graph.
* Serialization (Aligned Quads):
* Occurrence: (Context, Object, Concept, Value);
* Occurring: (Object, Context, Concept, Value);
* (Context : Concept type / label, Object, Concept, Value) DOM Property Graph.
* (Object, Context : Concept type / label, Concept, Value) DOM Property Graph.
* Layers APIs: Connector Bus. Templates, Forms Meta Model Dataflow
* Encoding:
* Object Model:
* Serialization (Aligned Quads):
* (URN, Context, Kind, Resource);
* Layer APIs: Addressing, Matching, Persistence.
* Augmentation:
* Object Model:
* Sets CSPO Model.
* Serialization (Aligned Quads):
* Sets CSPO Model Statements encoding.
* Layer APIs: Aggregation, Alignment, Activation. Sets Functional Dataflow Augmentations.
* Data (Data): key / value. Column: (price: 100). Data Aggregation Augmentation.
* Information (Schema): Record (keys / values relation): (price: 100, brand: ACME). Schema Alignment Augmentation.
* Knowledge (Behavior): Records (columns values relationship). Price variation behavior example: ((price: 100, brand: ACME, date: today, priceVariation: 0), (price: 110, brand: ACME, date: yesterday, priceVariation: 10)). Activation Augmentation: materialize relationships / facts.
* DDD REST HATEOAS DOM:
* Object Model:
* ID: Object Occurrence.
* Object (ID, Type, Member\*); Node.
* Type : Object;
* Member : Object; Arc (Property Graph).
* Serialization (Aligned Quads):
* (Object, ID, Type, Member\*);
* Layer APIs: Naming, Registry, Index. DOM Functional Dataflow.
* Messages: Dynamic Object Model Functional Monads bound (kinds signatures subscriptions) Functions. Contexts (Data, Schema, Behavior) browse traversal / transform. Resource aggregates Messages.
* (Resource, Transform, Mapping, Statement);
* Connector Bus API. Messages.
* Layers Dataflow Layout. Messages.
* Templates: Activation. Messages.