Objectives

Dados los mecanismos actuales de manejo de información y de la gestión de los procesos asociados a dicha informacion se pretende proveer a las herramientas actuales de medios aplicativos de la llamada gestión de bases de conocimiento que brinden insights en tiempo real tanto de análisis como de explotación de datos que ayuden a enriquecer con mejoras tanto la utilización del conocimiento como la toma de decisiones.

Description

Un enfoque sería el de “enriquecer” aplicaciones o servicios actuales con conocimiento relacionado al dominio de los mismos en el contexto de una interacción.

El otro podría ser “relevar” servicios y orígenes existentes para integrarlos en el contexto de un despliegue que convenga en exponer toda su funcionalidad actual aumentada y mejorada con servicios de bases de conocimiento y la integración declarativa con otros dominios o procesos.

Business Domain Translation of Problem Spaces

Un ejemplo del segundo punto anterior sería que las instancias de determinados casos de uso en el contexto del dominio de determinada aplicación “disparen” instancias de flujos de casos de uso en aplicaciones de diversos dominios cuya realización está relacionada en algún modo con la realización del primero.

Infer business domain process semantics and operations / behavior from schema and data (and services). Aggregate events, rules, flows.

P2P Purpose and capabilities discovery driven domain translation of business problem spaces. Enterprise bus of pluggable ontology domains, topics and peers providing features as backends (Big Data), alignments, rules, workflows, inference, learning and endpoints.

Architecture: logical view

Peers con Bundles desplegados resuelven y proveen los bindings necesarios (i.e.: persistencia, endpoints) a los recursos configurados declarativamente en los mismos (servicios, nodos, etc.) para que un Port (protocolo) resource pueda proveer a un Binding de plataforma el esquema y los datos necesarios.

Un Bundle puede agregar y mergear diversos orígenes de datos y servicios de aplicación, proveyendo "features" pluggables como descubrimiento e inferencia y el cliente de dicho Binding puede consumir en una interface uniforme los datos, esquema y comportamiento agregados desde los sistemas originales.

Resources: Functional implementation of diverse metamodel 'backends'.

Nodes: Metamodel instances for diverse Resource kinds, Resource specific aggregation and functional API.

Bundles: Declarative arrangement of Nodes and Resources.

Event driven dataflow between Resource, Node and Bundle (export activation signatures).

Resource type examples: RDBMSResource / DatasourceResource, AlignmentResource, PortResource / EndpointResource.

Events: Signatures / Instances. Resource, input, feature, output. Dimensional aggregated models (travel distance example).

Persistence event: (datasource, entity, key, object);

Alignment event, resource instances: (matcher, instance, context, instance);

Endpoint event: (endpoint, request, state, response);

Events instances only matches corresponding destination signatures.

Architecture: implementation view

Resource: (Name, Input, Feature, Output);

Node: Resource kind. Metamodel. Functional interface.

Bundle: Declarative arrangement of Nodes.

Node roles (by their Resource kind):

Backend / persistence:

- Inputs
- Features
- Outputs

Alignment / augmentation:

- Inputs
- Features
- Outputs

Port / endpoint (CRUD / behavior flows):

- Inputs
- Features
- Outputs

Binding: Client platform endpoints

- Inputs
- Features
- Outputs

Bundle: Declarative Resource Templates

- Internal declarative Metamodel representing deployment Resources. Reactive dataflow activation graph (distributed).
- Services, Nodes, Peers, Ports, Bindings Resource declarations. Subscriptions / transforms.

Node: Functional API / Metamodel. Resources: Metamodel, API activation signature. Node roles: Node Resource implementations: persistence, protocol, IO, alignment, peer, service, aggregation, etc.

Bundles: declarative resources / node bindings (bundle metamodel). Bundle resource implementation role (wraps nodes).

Resource interface IO: message dispatch, routes, endpoints, content type / format / contextual resolution of consumer (from subscription patterns). Protocols (quad / REST HATEOAS). Dialog message augmentation.

Message routing via Resource activation signature (resource, in, feature, out) pattern. Message IO coming from inner / outer Resource layers.

Resource resolution: index, naming, registry. Patterns. Templates.

Resources: Each Resource has its metamodel / functional / dataflow endpoint / interface (templates). Implemented reactive behavior according role (Service: persistence / alignment, Node: merge / augment, etc.): patterns & templates, IO (Resource functional implementation). Declarative Bundle description metamodel: instances and bindings of Resource(s).

Client platform bindings (augment services dialogs via events API over CRUD). Query client contexts over augmented state regarding schema / data of facts, info, knowledge. Common API: standard displaying / protocol (activation). JAX-RS, JAX-WS, JCA.

Example: Persistence Service over Apache Metamodel / JBoss Teiid via D2RQ. Node binding service links federated deployments. Port / Binding Resources expose services through protocol spec + endpoint service (IO).

Metamodel encoding: TensorFlow models. Aggregation. Layers. Reactive / Functional Node API.

Java platform binding: JCA / JavaBeans Activation Framework / XML Beans serialization (DataContentHandlers over standard generic model bean: REST / functional transform verbs over content type). XML / JSON HAL bindings. Export schema for DCI / ORM like bindings.

Bundles deployed as Apache ServiceMix / Red Hat Fuse OSGi bundles.

Implement ServiceMix OSGi Blueprint DSL (namespace) for Node, Metamodel and Bundle bindings implementations. Message subscriptions and pattern routing. Aggregation and Resource backends.

Blueprints archetype: Node metamodel impls. (backends, alignment, endpoint). Aggregation. Camel (reactive) routes: bindings / subscriptions, topics. Transforms. CXF (reactive JAX-RS) endpoint for endpoint metamodel resource monad. Client platform: JCA over endpoint protocol. Streams ETL. Request backend for specific patterns.

Metamodel: (Resource, Occurrence, Attribute, Value);
Metamodel: (Statement, Resource, Attribute, Value);
Metamodel: (Model, Statement, Attribute, Value);

Metamodel: backend Resource: IO (messages, persistence) / aggregate.

Resource API:
Resource.resource(String URI);
Static / factory. URI: Resource monad backend (JDBC, REST, SPARQL, etc.) Resource listens to / publish to. Apache Jena persistence interceptor / cache.

Resource (monadic instance) wraps their occurrence instances sets (occurrences, query, apply transforms).

Resource.occurrences() : Statement.

Statement.occurrences() : Model.

Metamodel aggregation hierarchies subclass monad wraps superclass instances. Flow : Rule : Event : Class : Kind : Fact : Model.

Resource.apply(Resource pattern) : Resource. Transform. Update. Apply pattern query / match: add / modify corresponding occurrences to player context resource.

Resource.query(Resource pattern); Apply to Model. Quad pattern matches. If none then build Resource from monadic resource factory. Performs resource activation (messages transform results, apply occurrences).

Metamodel messages: (match, apply) CSPO quads for each Resource hierarchy new instance: quads message. Apply occurrences to each local matching CSPO. Context of each applied CSPO: complement triple (i.e.: CPO for S) resources history. Metamodels aggregate new occurrences.

Resource history: invoked (match, apply) transforms in contexts until base resources. Complement based ID encoding.

Alignment: Semiotic contexts (sign, concept, object). Metamodel. Layers.

Domain Use Case

Domain, use cases: Music & Movies (plus DBPedia) retail, record, artist / publisher frontends. Core business cases plus enhancements. Integration with existing APIs.

Music / Movie store (buy, rental). BBC Music, IMDB, DBPedia, Geo / Mondial DB, Store DB (accounts, transactions, etc.). Amazon, iTunes, Netflix, Spotify, etc. Endpoints. Alignment (catalog abstract resource / concrete item resource: roles in context, ID, attributes in transactions / browsing). Platform endpoint (JavaEE JCA / REST HATEOAS HAL / JSONLD protocol nodes).

Features

linkeddata.org / Freebase / DBPedia (async) augmented. Time, places, etc.

Visualización

Visualization: Messages, Resources. Nested (context) tiles. Knowledge interfaces (activation operations).

UX: ZK / ZUL Templates & transforms from endpoints schema metadata / instances (tiles). JCA / JAF / DCI / REST. Activation domain browser.

Metamodel

Metamodel: Node instance specific Model monad endpoint (backend / triplestore sync / aggregation) wrapped. Specific Statement and Resource implementations. Bundle declarative settings. Model Statement / Resource Functional / CRUD: aggregate one object instance per each triple store quad.

Hierarchy: Model : Statement : Resource.

Resource(T extends Source): Quad. Example: DatabaseResource.

Statement(T extends Resource): Quad.

Model(T extends Statement): (Endpoint, Rsrc, Rsrc, Stmt);

Model: (Database / Service, Table / Op, Row / Args, Resource);

Aggregated resources: extends Model. Example: Fact(T extends Statement / FactStatement);

Metamodel: Apache Jena backed triple store for Model. Aggregated Resources: instantiated from Model (Model hierarchy).

Metamodel: Aggregation: Fact, types (dimensional), CEP. Rules. Flows. Streams (aggregate, align, reason).

Statement / Table: (Resource, Resource PK, Resource Col, Resource Val);

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

Model / Resource / Statement monads wraps specific node roles implementations of IO (specific models, Database, has specific resource and specific statement instantiated). Provides sources CSPO IO / CRUD. Then aggregates Fact, Kind, Class, Event, Rule, Flow (Resource monads wrappers of their players).

Fact (dimensionally aggregated from SPO): (Subject, Table, Column, Value);

Kind / Table: (Table, Subject, Column, Class);

Class: (Class, Table, Column, Value);

Event: (Event, Fact, Kind, Class); Fact occurring.

Rule: (Rule, Event, Kind, Flow); Aggregated from Events.

Flow: (Flow, Rule, Class, Class); Resulting attribute class flows.

Application / Binding: (Binding, Input, Match, Output); Bound functions.

Statement types wraps player resources. Resource wraps player aggregated statements.

Apache ServiceMix / JBoss Fuse implementation

Factories: Apache Camel custom component: “metamodel:nodeType” like namespaces. Pipes (Resource hierarchy): Blueprint Camel contexts. Prefixes for each InOut endpoint, resource hierarchy: metamodel:fact, metamodel:kind, etc.

Metamodels: Ontology (Apache Jena backed) service implementations (for each type of backend). Aligned / augmented / aggregated repositories / registry for Factory Message exchanges. Example: DB / Service Backend. MetamodelService provides features. Service binding from route contexts. Archetypes for development.

Pipes (Message IO) between each Resource hierarchy layers. Factory contexts activate on inputs over Metamodel service. Index service. Routing (dynamic, languages) Aggregators. Transforms (via dynamic resource in context plus template): enrich / filter (ID alignment), normalize (attribute / link alignment), sort (context alignment).

Message (hier parent):
MsgID: URI (history).
Headers: CSPO URIs.
Body: DOM Document (deep / rel links). Parsed for parents, occurrences IO in subclasses.
Attachment: Message representation.

Metamodel: Resource(Message) mapping: URI: MsgID, CSPO: Headers, Parent, occurrences: parse body / aggregate, transforms pipes for CSPO, factory. Representation: Type handlers for activation, REST command maps from metamodel metadata.

Resource API:
Resource(Backend, T extends Resource):
getParent() : T
getOccurrences() : T super Resource
getFactory() : Context
retrieve(CSPO pattern)
from(pattern parent, pattern occurs) add / set parent
occurrences(pattern parent?) : subcls
match(parent, pattern)
apply(occurs ctx, pattern newOccur) : adds occurrence
history(pattern parent)

Implement functional methods via Message transforms (XSL Templates, XPath, XQuery).

Metamodel: Message(Backend); Message / Backend IO. Backend occurs: Messages. Backend parent: Endpoint / Connection.

Resource arg: Super class.
Resource occurrences: Sub classes.

Resource(Backend) : Message
Statement(Resource)
Model(Statement)
Fact(Model)
Kind(Fact)
Class(Kind)
Event(Class)
Rule(Event)
Flow(Rule)

Metamodel service: Export hierarchy interfaces. Flow occurrences: Message / Resource.

Example context:
from=”metamodel:fact”
to=”metamodel:kind”
InOut endpoints / route. Pub / Sub producer / consumer Rx / async.

Transforms: Custom component pipes (metamodel namespace) routes for resource hierarchy and custom metamodel service implementation. Enrich (aggregate), filter (ID), normalize (attrs.) and sort (ctxs.) in rel to ctx resource via functional Resource API.

Dialog example (fact / kind):

1. Fact - Kind (Fact w/o Kind aggregated)
2. Retrieve Kind occurrences in Fact (pattern) context (existing / created classes)
3. Create / retrieve matching Kind. Apply Kind to matching Facts (Kind occurs).

Component URI invocation example of Resource API:
metamodel:kind[selector]:fun(args / patterns)

References

SemanticWeb. OWL. RDF:
http://infomesh.net/2001/swintro/
http://www.linkeddatatools.com/semantic-web-basics
https://www.w3.org/2013/data/

Big Data / Big Linked Data / LDP: https://en.m.wikipedia.org/wiki/Linked_Data_Platform
https://project-hobbit.eu/tag/big-linked-data/

Actor / Role pattern, DCI, Functional programming:
http://www.cs.sjsu.edu/~pearce/oom/patterns/analysis/Actor.htm
https://en.m.wikipedia.org/wiki/Data,_context_and_interaction
https://dzone.com/articles/functional-programming-in-java-8-part-0-motivation
http://www.nurkiewicz.com/2016/06/functor-and-monad-examples-in-plain-java.html?m=1

Reactive programming. Event driven architecture. Dataflow:
https://gist.github.com/staltz/868e7e9bc2a7b8c1f754
https://en.m.wikipedia.org/wiki/Dataflow

ESB / EAI:
Apache ServiceMix.
Red Hat Fuse.

Integración / persistencia:
Apache Metamodel.
JBoss Teiid.

Java platform binding:
JCA (Java Connector Architecture).
JAF (JavaBeans Activation Framework).
Beans Serialization API.
DCI / REST (HATEOAS / HAL).