Business domain translation of problem spaces (WIP draft)

Introduction

Keywords: Semantic Web, RDF, OWL, Big Data, Big Linked Data, Dataflow, Event Driven, Message Driven, ESB, EAI.

Objectives: Streamline and augment analysis and knowledge discovery capabilities for enhanced declarative and reactive event driven process flows between frameworks, protocols and tools for Semantic Web backed integration and Big (linked) Data applications.

Achieve diverse information schema merge and interoperability (for example for different domains or applications databases). Translate behavior in one context (domain) into corresponding behavior in other context. Aggregate diverse domain data (facts) into corresponding domains state.

Feature: Data backends / services virtualization (federation / syndicaton / synchronization)

Any datasources (applications / services) entities and schema regarded as being meaningful for a business domain translation and integration use case, regardless of their source format or protocol, should be provided of an adapter mechanism (Node + Service) which makes it available as ‘reactive endpoint’ which provides CRUD services for other Node / Service to consume / synchronise.

Feature: Schema (ontology) merge / match / alignment

Diverse domain application data with disparate backend databases and services and diverse sources of business data (linked data ontologies, customers, product and suppliers among others) are to be aligned by merging matching entities and schema, once syndication and synchronization are available.

Examples: different names for the same entity, entity class or entity attribute.

Feature: Domain translation of problem spaces

Object model: schema / data of facts, information, knowledge (behavior).

Given a business domain problem space (data, schema and behavior for an application, for example) a ‘translation’ could be made between other domain(s) problem spaces which

encompasses a given set of data, schema and behavior instances (objectives) to be solved in the other domains in respect of the problems in the first domain.

This shall be accomplished by means of an event-driven architecture over a semantics aware layer which leverages diverse backend integration (sync) and schema merge / interoperability. Also, a declarative layer is provided for aggregation and composition of related event flows of various objectives to met a given purpose (dataflow activation).

Example: In the healthcare domain an event: flu diagnose growth above normal limits is to be translated in the financial domain (maybe of a government or an institution) as an increase of money amount dedicated to flu prevention or treatment. In the media or advertisement domain the objectives of informing population about flu prevention and related campaigns may be raised. And in the technology sector the tasks of analyzing and summarizing statistical data for better campaigns must be done.

Deployment: Node abstraction

A Node is a Service (Node interface extends those of Service). Service is available for discovery and binding. Node(s) and Service(s) are message-oriented and become ‘activated’ (dataflow) when they can provide an ‘enhancement’ over a pertaining subject or entity within dialogs (messages) in their scopes.

Bindings: Patterns / Templates. A Node has a collection of declarative bindings (dataflow subscriptions). Binding Templates specify patterns for the declarative activation of Services to be bound (scope, instantiation, activation: data / schema of facts, information and knowledge / behavior).

Discovery (orchestration): Service instantiation / activation. Scopes. An application (context) deployment is comprised of one or more Node(s) which depends declaratively on other Node(s) and Service(s) for fulfilling its behavior augmentating or being augmented (enhancements) regarding knowledge schema and behavior flows.

Deployment: Service abstraction

Services (and Nodes exposing Services) are declaratively specified for its scope having data / schema metadata for its:

Interfaces: Endpoints.
Messages: Object Model.

A Service is a collection of Message : (Facts, Information, Knowledge schema / behavior) definitions for the arguments / return values for its Endpoint interfaces.

Deployment

Service and Node bindings are the means of declaratively composing application behavior. Services may include, for instance, a metamodel service suitable for aggregation and inferencing over statement data. Services may also include, for example, an index or a registry service used, perhaps by other services, for functional composition of behavior.

Discovery mechanisms resolve proper Service instances to be bound into Node flows. A Node Service may be requesting by a given Service in a given scope which may be, for example, ‘singleton’ (one Service instance per binding) or have a compound (shared) scope. A Node would have, for example, its own metamodel service instance but may share a scoped registry service with another Node(s). Scoping, nesting.

Client: Protocol / platform bindings

Protocol bindings (Node IO) Services:

REST HATEOAS (HAL / JSONLD). LDP.
SOAP.
OData.
SPARQL.

Internal Node(s) knowledge and Services exposed (by some Node IO) via these protocols. In the case of REST / LDP Resources endpoints implemented HTTP methods are created from knowledge (aggregated, aligned and augmented) entity classes and instances (for CRUD). For SOAP and OData a similar ‘object model’ is exposed via corresponding interfaces.

SPARQL May mix ‘high level’ aggregated schema and instances with lower level RDF / OWL statements or metadata.

Interactions (knowledge / behavior) is to be modelled also as ‘objects’ into the ontology (for all protocols) and, for example, what would be otherwise stated as a ‘method’ in some platform is rendered as a class of some entity to instantiate and for which populate ‘roles’ and whose results are functional relationships via the ‘activation’ of other resources.

Platform bindings (Clients) Services:

JavaEE (JPA / JVM dynamic language).
JavaScript (browser).
JavaScript (NodeJS).
PHP.
.Net (LINQ).

Platforms bindings are meant to provide a ‘native’ language / platform representation (classes and instances) of knowledge (data, schema and behavior) stored into Node(s) via their services.

Interaction with knowledge aware nodes is provided by platform bindings wrapping calls into an specific protocol binding. Then, entities (classes / instances / behavior) of each specific platform / language / runtime are ‘generated’ from parsed data and metadata.

This way business applications of different platforms in different languages will leverage the benefits of having a real time integration and interoperability backend.