The Vocabulary Hub to configure data space connectors

Wouter.vandenBerg@tno.nl
Michiel.Stornebrink@tno.nl
Arjan.Stoter@tno.nl
Jan_Pieter.Wijbenga@tno.nl

Introduction

Interoperability within a data space requires participants to be able to understand each other. But how do you get data space participants to use a common language? According to the IDS Reference Architecture Model (IDS-RAM)[1], the main responsibility for this common language lies with an intermediary role called a *vocabulary provider*. This party manages and offers vocabularies (ontologies, reference data models, schemata, etc.) that can be used to annotate and describe datasets and data services. The vocabularies can be stored in a *vocabulary hub*: a service that stores the vocabularies and enables collaborative governance of the vocabularies.

The IDS-RAM specifies little about how vocabularies, *vocabulary providers* and *vocabulary hubs* enable semantic interoperability. The hypothesis that we address in this position paper is that a *vocabulary hub* should go a step further than publishing and managing vocabularies, and include features that improve ease of vocabulary use. We propose a wizard-like approach for data space connector configuration, where data consumers and data providers are guided through a sequence of steps to generate the specifications of their data space connectors, based on the shared vocabularies in the *vocabulary hub*. We illustrate this with our own implementation of a *vocabulary hub*, called Semantic Treehouse.

Background

Semantic interoperability is vital for a data space. According to a recent Open DEI publication[2], the design and implementation of a data space comprises a number of functional building blocks. One of the categories in the Open DEI taxonomy describing those building blocks is 'interoperability' including data exchange APIs, data representation formats and data provenance and traceability. Furthermore, as the European Interoperability Framework[3] shows, interoperability itself unfolds into different interoperability challenges, one of them being semantic interoperability.

The importance of semantics in data spaces is also clearly reflected by the work done by the International Data Spaces (IDS) initiative. One of the few normative specifications of IDS is the IDS Information Model[4]. As such, it should be considered as a prescriptive part of the IDS Reference Architecture Model (IDS-RAM). A common language is necessary for the level of semantic

[1] IDSA Reference Architecture Model 3.0: https://internationaldataspaces.org/use/reference-architecture/
[2] OpenDEI Position Paper: Design Principles for Data Spaces, https://www.opendei.eu/
[3] https://joinup.ec.europa.eu/collection/nifo-national-interoperability-framework-observatory/3-interoperability-layers
[4] https://github.com/International-Data-Spaces-Association/InformationModel

interoperability that is required to achieve the main design goal of IDS: (semi-) automated exchange of digital resources.

To semantically annotate the data that is being shared in a data space, we require domain specific vocabularies. The development of such vocabularies is often organized centrally by business communities and delegated to some sort of standards development organization (SDO) that publishes and maintains shared domain vocabularies and data schemata. In the IDS-RAM, this role is called the *vocabulary provider*, and the platform where communities publish and maintain shared vocabularies is called the *vocabulary hub*.

A short overview of Semantic Treehouse

TNO's implementation of a *vocabulary hub* is called Semantic Treehouse. It is an online community platform for data models and vocabularies and gives *vocabulary providers* the tools they need to facilitate semantic interoperability in their data space. Semantic Treehouse is a web application that can be hosted (e.g. by a *vocabulary provider*) to serve the needs of specific business communities, e.g. a logistic sector, or other domains of interest. It provides that community with core *vocabulary hub* functionality, including publishing and collaborative maintenance of vocabularies and data models.

For more than a decade, TNO has developed and maintained standardized semantic data models for a variety of industries[5]. Standardization activities include governance, modelling, maintenance, community management and implementation support. While using many different tools, communication channels and knowledge sources, the idea arose that these should be unified. Development of Semantic Treehouse started in 2015, at first as an internal tool to help us carry out our own activities more effectively. The main design goals were: (1) web-based publication of data models for easier viewing, (2) improved maintenance and knowledge management with GitHub-style issue tracking, (3) increased user community participation through transparent presentation of working group information, and (4) to offer implementation support through the provision of data validation services in multiple formats (e.g. JSON, XML).

Since 2015, Semantic Treehouse has evolved from a tool used within TNO to software provided as a service (SaaS) to SDOs in the Netherlands. One of these SDOs is the Smart Connected Supplier Network (SCSN) that provides an operational IDS data space for the smart industry sector[6][7]. Currently, Semantic Treehouse serves the needs of 750 unique business users per month, with more than 4000 registered accounts in total. This has validated our belief that Semantic Treehouse provides a strong foundation to facilitate SDOs tasked with developing and maintaining shared data models together with end users in their business communities.

TNO regularly releases new versions of Semantic Treehouse, where new features and improvements are a combination of ideas and requests from clients and strategic research topics. Here, we provide two examples of the latter. First, providing SaaS is an intermediate step to releasing Semantic Treehouse as open source, which we expect to achieve in 2023. Second, Semantic Treehouse is continuously adapted to new innovations and technology trends, one of which is a shift from

[5] For more info, see: https://www.tno.nl/en/focus-areas/information-communication-technology/roadmaps/efficiency-effectiveness-quality-and-the-costs-of-systems/scalable-it-systems/standardisation-of-semantics/
[6] https://internationaldataspaces.org/adopt/data-space-radar/
[7] https://smart-connected.nl/

hierarchical message models towards ontologies and applying FAIR principles. This shift has shown that there is still a lot that can be done to boost semantic interoperability in business communities.

As a means to facilitate semantic interoperability, a wizard-like component was integrated in the *vocabulary hub* to further drive adoption of shared vocabularies. The wizard takes a user through a series of steps to design message schemas and API specifications based on a shared domain ontology. The wizard generates schemata for JSON, XML and CSV data formats, generates example messages and generates mappings for automatic transformations of data to RDF according to the ontology. The generated JSON schemas can be used directly in OpenAPI specifications to configure a data space connector. With this functionality we aim to bridge the gap between semantic web technologies and the traditional world of IT development, and bring semantic interoperability closer to data spaces.

The data space connector configuration wizard

Architecture

Data exchange requires establishment of its contents. The design principle 'separation of concerns' implies that, given the availability of a single semantic model, i.e., the ontology, the establishment of the contents can be done on the level of semantics without concern on the syntactical forms. The semantic model allows to cherry pick from the graph tree those elements that constitute the information that is to be exchanged. This results in a set of one or more sub-graphs (with selected sub-elements only) from the ontology, together denoted as the 'abstract message'.

The wizard component allows users to 'cherry pick' the relevant classes and properties from the ontology in the following way. In the first step, the user selects the information that is to be exchanged, which the wizard collects into an abstract message tree (AMT); In the second step, the wizard generates a technology-specific syntax binding between the AMT and a syntax format of the user's choice, e.g. XML or JSON; Finally, in the third step, the wizard generates the specifications for a conversion between message- and/or query-oriented data exchange using RML[8]. The use of the resulting RML is optional (Figure 1).

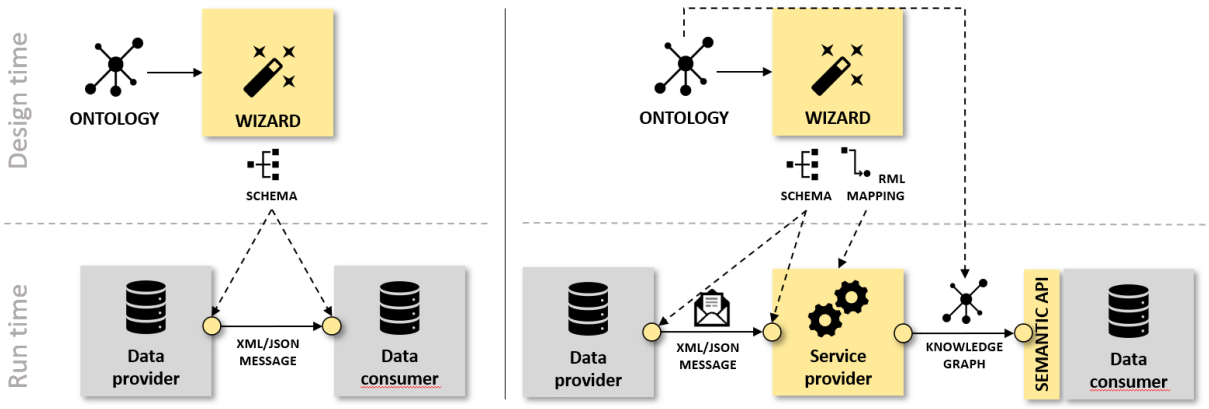

FIGURE 1 - (A) SYNTAX BINDINGS IN A SINGLE STACK, (B) SYNTAX BINDINGS AND RML MAPPINGS IN A DUAL STACK

In the first scenario (Figure 1a), the output of the wizard is the generated schema sourced from the ontology. In the second scenario (Figure 1b), the wizard-generated RML mapping is also used.

[8] https://rml.io/

We introduced an RML Engine in order to transcribe between message and graph-based data in accordance to the generated mapping specification (RML). Because applications are independently developed without agreements on their data structures in advance, we apply graphs as structure-agnostic data representation to allow mediation between distinct message structures. Similarly, the semantic API can be configured to transcribe from graph data to the native data schemata of communicating peers. As such it mirrors the operation of the RML Engine.

Separating the two components anticipates on future transition phases that allow for the inclusion of SPARQL endpoints.

Procedure

The procedural design of the wizard assumes one or more ontologies as input and ends with the schema and RML specifications as output.

Step 0 - Exploration of the ontology: Assume a domain ontology. Exploring and familiarizing oneself with the ontology is required before starting the Wizard. The available WebVOWL[9] and LODE[10] tools are two examples of how the domain user can get acquainted with the semantics that apply.

Step 1 - Message composition: This is performed in two sub-steps:

Step 1.1 - Message Model Set-up: The wizard is offered in a collaborative environment (the *vocabulary hub*) with a shared workspace for standardized specifications, and a private workspace where each community stakeholder manages its own specifications. The versioned specifications come in two types: ontologies and abstract message models. Every registered user or organization can create their own project that groups private and shared specifications and a sandbox for tryouts. Shared projects exist that contain the agreed-upon specifications. Typically, the domain ontology lives in a shared project and is used by all domain stakeholders throughout the message specification process.

The wizard creates or chooses a project that in its basic form consists of a title, description, versions of a message model, with status information (concept, final) and other metadata (dates, documentation, acknowledgements). Namespaces consolidate the universal uniqueness of the generated output schemata. Each message model imports at least one versioned ontology, presented as graph trees for the subsequent message composition.

Step 1.2 - Select root class: After the above initial set-up, the user selects the ontology class that acts as root to the message model. The default values for name, URI, type and comments are loaded from the ontology and populates the root element. The user can then alter these values, e.g., element names.

Step 2 - Customize root class and iteratively select and customize properties: The incoming and outgoing properties of the root class are loaded from the ontology and are all presented as potential elements that can be added as child elements of the message root element. Allowed customizations include renaming and adding further restrictions to the default ontological specifications into a customized message specification. For example, changing the cardinality of some property from [0..n] to [1..n] counts as a further restriction. To prevent cardinality violation of the ontology, the wizard disallows loosening constraints, e.g., changing the cardinality from [0..1] to [0..n]. The wizard also allows structural changes, e.g., adding a property more than once or inserting a grouping element. Cardinalities are always checked against violating the ontological constraints.

This step can be repeated for each element that sources from the root element, further unfolding the graph until all desired information is included in the message model. Note that

[9] http://vowl.visualdataweb.org/webvowl.html

[10] https://essepuntato.it/lode/

'walking back' over properties, i.e. going from range to domain, is only supported for object properties. This requires to include a wizard-specific annotation.
The result of this step is the abstract message schema that holds all necessary information in an agreed structure (element names, cardinalities, URIs, property mappings, sequences).

Step 3 - Generate and validate the specifications: Based on the abstract message schema, the specifications that act as configurations for the RML Engine can be generated by a press of the button. This consolidates a message model version for all specifications: XML or JSON Schema, the RML mapping, and example data as follows:

Step 3.1 - XSD Generation: With the target and message namespaces, the global complex types and simple types are determined according to the Venetian Blind strategy[11]. Next the root element, global complex and simple types are added, followed by the namespace prefixes.

Step 3.2 - JSON Schema Generation: The JSON Schema is generated in accordance with the json schema draft 07 version. The mapping of simple types was inspired by the OxygenXML mapping[12].

Step 3.3 - RML Generation: What RML to generate depends on the choice for JSON or XSD. An RML logical source is created for the target, and a subject map is created for the root element. For leaf nodes, a simple RML predicate object mapping is created. For nodes that have sub-elements, a nested triples map is created that is bound to the parent through a logical join condition. This is done recursively until all nodes have been processed.

Based on test data, the data messages and their conversions can be syntactically verified against the specifications that apply.

Conclusion

In this position paper we stipulated the importance of semantic interoperability according to common reference and interoperability frameworks. We explained how SDOs enable standardized data exchange in business communities by means of vocabularies and elaborated on the need for tooling to support SDOs and end users to design, publish, share and maintain those vocabularies. Furthermore we specified how a wizard-like component can be used to design ontology based data exchange schemata and, if needed, data transformation specification that can be directly used to configure a data space connector. Thereby bringing semantic interoperability to data spaces, which enables non-technical domain experts, i.e. people without training or experience in knowledge representation, to collaboratively design the required APIs in a matter of days.

Acknowledgements

The work as presented in this paper builds upon the work done within two projects. One is the Dutch research project *Flexible IT with Ontologies*, supported by the Dutch Top consortia for Knowledge and Innovation Institute for High Tech Systems and Materials[13] of the Ministry of Economic Affairs and Climate Policy. The other is the BD4NRG project, which is co-funded[14] by the Horizon 2020 Programme of the European Union (grant agreement No 872613).

[11] http://www.xfront.com/GlobalVersusLocal.pdf
[12] https://www.oxygenxml.com/doc/versions/22.0/ug-editor/topics/xsd-to-json-schema-converter.html
[13] https://hollandhightech.nl/
[14] This document reflects only authors' views. The European Commission is not liable for any use that may be done of the information contained therein.