SOyA 🌱 - Semantic Overlay Architecture

White Paper

Authors

- Fajar J. Ekaputra (fajar.ekaputra@tuwien.ac.at)
- Christoph Fabianek (christoph@ownyourdata.eu)
- Gabriel Unterholzer (gabriel.unterholzer@dec112.at)

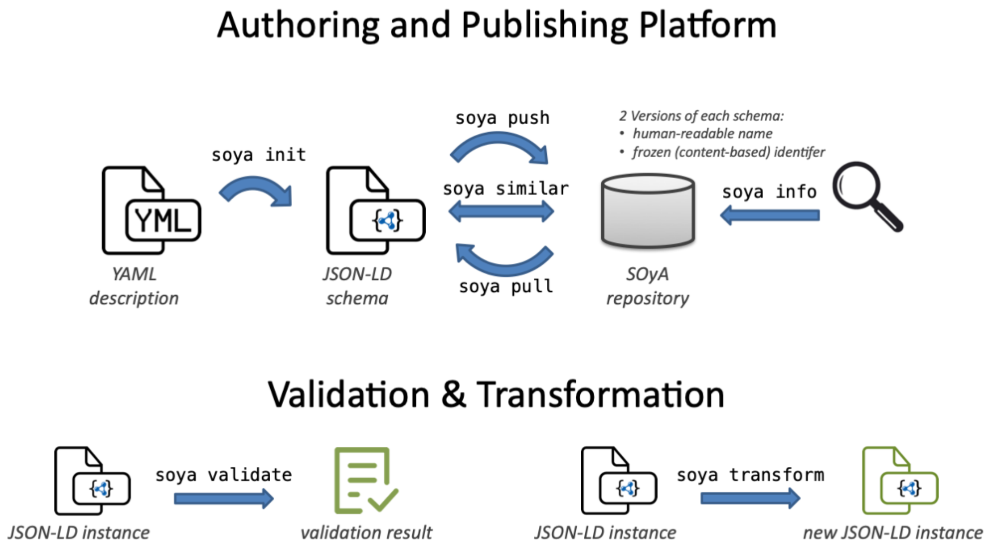
Status

- draft, November 22nd, 2021

Content

Introduction

SOyA is a data model authoring and publishing platform and also provides functionalities for validation and transformation. It includes a command line tool and an online repository for hosting these data models.



A data model (schema) consists of one or more base structures holding a list of attributes with respective data types. A base can be semantically enriched with overlays that describe further details about character encoding, classification for specific purposes, or constraints for validation and more. Bases and overlays may be authored in YAML and are eventually represented in JSON-LD v1.1 to leverage existing Linked Data tools.

SOyA objects (bases and overlays) are published to an online repository and become accessible through a defined human-readable name as well as through a content-based identifier (a fingerprint unique to each SOyA object). In case of updates to the original document, the original version is still accessible through this content-based identifier.

This white paper provides an overview about the current development state of the Semantic Overlay Architecture. You can read about goals and definitions used during developing SOyA or jump right into the tutorial to try the available tools.

All interested parties are welcome to provide feedback and ask questions using the Github issue tracker: https://github.com/OwnYourData/soya/issues

Objectives

Stakeholders and their User Stories

This section provides a list of relevant stakeholders/actors in the life-cycle of a data model and of instances using those data models. Together with a short description of each actor it lists requirements in the form of user stories to describe their specific needs.

Solution designer
A person that designs schemas & overlays as well as oversees how it fits into an overall solution.

- As a solution designer I want to easily describe the structure and properties of a dataset (without prior semantic knowledge) to create a semantically well-defined model.
  → achieved through the use of YML and soya-cli

- As a solution designer I want to annotate attributes in a data structure (e.g., what is PII in a record, how to define subsets of a record, etc.)
  → achieved through the use of overlays

- As a solution designer I want to browse / contribute to / extend existing definitions.
  → achieved through the soya repository and soya-cli

- As a solution designer I want to be able to use decentralized technologies.
  → achieved through the use of DRIs (as a back-stage feature handled in the soya repository)

Storage provider
A person or organization that hosts repositories for schema definitions and/or databases with records using those definitions (e.g., http://w3id.org or Semantic Container hosting).

- As a storage provider I want to efficiently manage data; this includes simple handling of CRUD operations as well as flexibility in using backend database solutions.
  → achieved through soya repository and reference implementation in Semantic Container

Data engineer
A person that sets up the infrastructure defined by a solution architect; this can include setting up a local SOyA registry, starting and configuring Semantic Containers, or simply verifying availability of relevant linked data sources.

- As a data engineer I want to validate a given dataset against a model.
  → achieved through validation overlay and soya-cli

- As a data engineer I want to map legacy datasets to a given data structure represented in SOyA.
  → achieved through transformation overlay and soya-cli

- As a data engineer I want to map a given SOyA data structure to a target standardized / canonical definition.
  → achieved through mapping and transformation overlay and soya-cli

- As a data engineer I want to transform a dataset between various formats and encodings.
  → achieved through encoding and transformation overlay and soya-cli

- As a data engineer I want to convert flat data (JSON, CSV, XML) into RDF to use the Semantic Web Stack and integrate with Linked Data.
  → achieved through acquisition overlay and soya-cli

End user
A person that collects, views, and edits data described by / annotated with SOyA principles.

- As a user I want to render a form for display.
  → achieved through soya-render library

- As a (Semantic Container) user I want to edit records (JSON, CSV, XML, RDF) in a form.
  → achieved through soya-render library

- As a user I want to easily reference existing data.
  → achieved through design decision for JSON-LD

Key Characteristics

- **Open:** all components are open source, free to use (incl. commercially), and publicly accessible (Github, public Repository)
- **Extensible:** design is inherently supposed to be extended through own definitions, extensions, concepts
- **Compatible:** allow seamlessly switching between data formats to use the best technology for the given use case
- **Ease of use:** make it as simple as possible (but not simpler!) through documentation (e.g., tutorials, examples), authoring tools (e.g., YAML for initial description), UI components (web form), and integration (e.g., Semantic Containers)
- **Focus on Semantic Web:** build on top of and make use of the Semantic Web Stack
- **Decentralised:** avoid any centralized components or addressing (i.e., use decentralized resource identifiers - DRIs - where possible)

Roadmap

This section describes possible next steps in the development in SOyA. Anybody should feel free to add ideas here and also comment on existing proposals. When certain elements materialize they are moved to the respective sections.

- describe existing and well-known ontologies (e.g., foaf) through SOyA; this might provide insights from mature data structures and could spur interest from a wider community
- implement acquisition overlay type to transform flat data into RDF

- integrate SOyA into Semantic Container to demonstrate validation and transformation mechanisms in a data flow
- rendering of forms to visualize datasets and allow to create & edit records in an easy to use manner

Definitions & Structure

Base

Holds the following information:
- name
- list of attribute names and associated type

A base layer is identified through a name and a DRI calculated from its canonical form when represented in JSON-LD v1.1 (see DRI calculation).

Set of Bases

Multiple bases can be combined in a single definition for related concepts.

A set of bases can be identified through a name and a DRI calculated from its canonical form when represented in JSON-LD v1.1 (see DRI calculation).

Overlay

The following types of overlays are defined in the default context (https://ns.ownyourdata.eu/soya/soya-context.json) and can be associated to a Base:

- **Acquisition**
  allow converting flat data into RDF

- **Alignment**
  reference existing RDF definitions (e.g. foaf); this also includes declaring a derived base so that attributes can be pre-filled from a data store holding a record with that base (e.g., don't require first name, last name to be entered but filled out automatically)

- **Annotation**
  translations for name and descriptions of schema base; labels and descriptions of attributes; supporting multiple languages

- **Classification**
  select a subset of the attributes (e.g., PII, listView)

- **Encoding**
  specify the character set encoding used in storing the data of an instance (e.g., UTF-8)

- **Format**
  defines how data is presented to the user
- **Transformation**
  define for each element in a structure how it should be transformed
- **Validation**
  predefined entries, range selection, any other validation incl. input methods

Additional overlay types can be generated by providing a custom context.

An overlay is either part of a base layer definition or can be a stand-alone structure and identified through a name and a DRI calculated from its canonical form when represented in JSON-LD v1.1 (see DRI calculation).

Stack

A stack combines one or more bases with zero to many overlays to be used in an instance.

A stack is identified through a name and a DRI calculated from its canonical form when represented in JSON-LD v1.1 (see DRI calculation).

Instance

An instance is the canonical representation of a concrete data record.

An instance is identified through a name and a DRI calculated from its canonical form when represented in JSON-LD v1.1 (see DRI calculation).

Structure

A structure is an object holding SOyA information and has a unique name within a repository. Additionally, there exists a DRI as a unique and location independent reference for this structure.

DRI

A DRI (Decentralized Resource Identifier) represents a content based address for any given JSON object. In the context of SOyA the DRI calculation adheres to the following steps to increase interoperability and decentralization:

1. expand JSON-LD to not include references in @context section
2. check all keys and values if they represent a DRI in a full URL and in that case only use the DRI
3. transform JSON into canonical form based on this standard (currently still in draft): https://tools.ietf.org/html/draft-rundgren-json-canonicalization-scheme-16
   I.e., order elements, flatten structure, and if necessary introduce "@id" for blank nodes
4. remove self-referencing entries required by JSON-LD:
   a. for layers remove the key/value pair with the key "@base" in "@context"
   b. for instances remove all key/value pairs with the key "@id" in "@graph"

5. calculate the hash value using Multihash[1]
   (default hash function as of 2021: SHA2-256)
6. encode the resulting binary from the hash value using Multibase[2]
   (default encoding as of 2021: base58btc)

An example implementation to calculate the DRI from an SOyA object in Ruby is available here: https://github.com/sem-con/sc-soya/blob/main/test/dri.rb

References to Instances

References can be either a URI or a blank node (i.e., does not have an ID).

a) Blank Node: a JSON object
   `{"key":"value", ...}`
b) URI: a reference in the form
   `{"@id":URL with DRI reference}`

Example: see Person Instance below for a list of employers with external class and instance mix

[1] https://github.com/multiformats/multihash
[2] https://github.com/multiformats/multibase

Tools

SOyA CLI

available at: https://github.com/OwnYourData/soya/tree/main/cli

<u>Functionalities</u>

- `soya template "type"`
  to create a yml template (type are entities defined in SOyA Ontology)
  - used for bootstrapping the YML file (for a solution designer)
- `cat name.yml | soya init`
  to create a valid JSON-LD v1.1 document (can be stored in `name.jsonld`)
  - convert YAML to JSON-LD (human-readable version)
  - includes validation (structural validation, YML validation)
- `cat name.jsonld | soya similar [--repo host]`
  to find similar base definitions; outputs list of references to base layers with similarity score (0-1)
  - to check the newly-minted JSON-LD → find similar "stack" with closely related
- `cat name.jsonld | soya push [--repo host]`
  to store content of `name.jsonld` in a SOyA repository hosted at `host` (default: soya.ownyourdata.eu)
  - creates DRI version for permanent availability
- `soya pull entity [--repo host]`
  to read entity definition from a SOyA repository
- `soya info entity`
  to list all entities that had specified name (only latest is accessible by name and other entities through their DRI)
- `cat data.jsonld | soya validate validation_overlay`
  to validate a given dataset (instance) against an overlay definition
- `cat data.jsonld | soya transform transform_overlay`
  to transform a given dataset (instance) based on transformation overlay definition (this includes first validating if data.jsonld validates against the base layer of transform_overlay)

SOyA Repository

available at: https://github.com/OwnYourData/soya/tree/main/repository

- Docker image: https://hub.docker.com/r/oydeu/soya-base

Functionalities

- read SOyA documents (for `soya pull`)
  URL: GET /name
  Examples:
  - `curl https://soya.data-container.net/Person`
- get info about SOyA documents (for `soya info`)
  URL: GET /name/info
- write SOyA documents (for `soya push`)
  URL: POST host/api/data (same as for SemCon, provide: name, document)
  2 possibilities:
  - if name is human-readable (i.e., not /zQm.{42}/):
    convert to DRI (`soya dri`) and store both version (i.e., the entry will be forever accessible via this DRI and as human-readable identifier as long as later nobody else is using this human-readable name)
  - if name is DRI (i.e. /zQm.{42}/):
    store only DRI version
- find similar SOyA documents
  URL: POST /api/similar (body: SOyA base layer document)
  response: array (paged, default is only first 20 entries) of all base layers with similarity measured ordered descending

Tutorial

This tutorial describes a simple workflow to model a data structure representing a person and managing data. It includes the tasks by a solution designer to describe the model, a data engineer managing records, and a storage provider hosting the data.

You can either install the SOyA CLI tool from Github (link) or use a pre-built Docker image with soya and required tools pre-installed (Docker image: oydeu/soya); use the following command to run the soya cli docker image:

```
docker run -it --rm oydeu/soya
```

Designing the Data Model

Start with a simple base layer (describing a person) provided as template:

```
$ soya template base
meta:
  name: Person
content:
  base:
       - name: Person
         attributes:
           name: String
           dateOfBirth: Date
           age: Integer
           sex: String
```

Transform the YML into JSON-LD using the SOyA CLI:

```
$ soya template base | soya init
{
  "@context": {
       "@version": 1.1,
       "@import": "https://ns.ownyourdata.eu/ns/soya-context.json",
       "@base": "https://soya.data-container.net/Person/"
  },
  "@graph": [...]
}
```

Find similar base definitions:

```
$ soya template base | soya init | soya similar
```

a list of SOyA objects available at the default repository https://soya.data-container.net

Write the base definition into the online repository:

```
$ soya template base | soya init | soya push
Person
```

you can show the Person definition in the repository now under https://soya.data-container.net/Person

Query available information about the Person SOyA object:
```
$ soya info Person
```
shows fingerprint (DRI), history, and available overlays for the object

You can also experiment with overlays by using available templates:

- show available templates
  ```
  $ soya template -h
  ```
- create an annotation overlay
  ```
  $ soya template annotation
  ```
- use the same commands as above to transform from YML to JSON-LD and publish:
  ```
  $ soya template annotation | soya init | soya push
  ```

Managing Records

SOyA currently supports validating JSON-LD instances of SOyA objects and transforming between different SOyA bases.

Validate if a given record conforms to an validation overlay:
```
$ curl -s https://playground.data-container.net/PersonValid | \
  soya validate Person_Test
```

(use `soya pull Person_Test` to show the validation overlay, and view the test record here: https://playground.data-container.net/PersonValid)

Transform records between base layers:
```
$ curl -s https://playground.data-container.net/PersonAinstances | \
  soya transform PersonB
```

Comparison

SOyA was influenced by a number of other technologies and this section should highlight those connections as well as what makes SOyA distinct.

Overlay Capture Architecture (OCA)

URL: https://oca.colossi.network/
Shared characteristics

- Base / Overlay nomenclature and structuring
- description of properties like PII and multi-language support
- DRIs are used to reference

Differences between OCA and SOyA

- SOyA describes Base (Schema Base) as a graph in RDF instead of flat list in OCA
- SOyA overlays are defined via RDF (to allow OWL2 reasoning) instead of a list of properties in OCA
- SOyA provide options to reference existing vocabularies and ontologies
- SOyA has additional overlays for validation and transformation

Layered Schemas

URL: https://layeredschemas.org/
Shared characteristics

- Base / Overlay nomenclature and structuring
- use of JSON-LD v1.1 to represent entities (is now focusing on labeled property graphs)

Differences between Layered Schemas and SOyA

- transformations are more advanced, structured, and powerful through Semantic Process Pipeline in Layered Schema
- Layered Schema uses human-readable names (or any string) to reference objects while SOyA pursues the strategy to store each object in the online repository twice: one instance uses the provided human-readable name and the other instance is referenced through an automatically generated DRI (i.e. a content-based address);the second instance therefore represents an immutable / frozen snapshot

References

- JSON-LD 1.1
- JSON-LD playground
- libraries that work with JSON-LD 1.1
- RML: https://github.com/sem-con/Tutorials/blob/master/4_SPARQL/README.md
- JSON transformation: https://github.com/bazaarvoice/jolt
- jq: https://stedolan.github.io/jq/ (Playground: https://jqplay.org, Ruby bindings: https://github.com/winebarrel/ruby-jq)