SMART: Towards Automated Mapping between Data Specifications

S. Kalwar*, M. Sadeghi*, A. Javadian Sabet*, A. Nemirovskiy* and M. Rossi†
* Dipartimento di Elettronica, Informazione e Bioingegneria
† Dipartimento di Meccanica
Politecnico di Milano
Piazza Leonardo da Vinci 32, I-20133 Milano, Italy
Email: firstname.lastname@polimi.it

***Abstract*—The ability to perform automated conversions between data conforming to different specifications is a key ingredient to achieve interoperability among heterogeneous systems—which, in turn, is at the basis of the creation of so-called Systems of Systems. These conversions require the definition of mappings between concepts of separate data specifications, which is typically a hard and time-consuming task. In this paper, we present a technique to automatically suggest mappings to users, based on both linguistic and structural similarities between terms. The approach has been implemented in our prototype tool, SMART (SPRINT Mapping & Annotation Recommendation Tool), and it has been validated through tests carried out using specifications from the transportation domain.**

***Keywords*—Ontology, linguistic similarity, structural similarity, automated mapping, natural language processing**

I. Introduction

In recent years there has been a growing interest in the development and deployment of so-called Systems of Systems [1], [2], where independent, heterogeneous systems built using different technologies interact to provide complex services. A paradigmatic example of this trend is found in the transportation domain, especially in the European Union, where initiatives are underway to create a Single European Transportation Area [3], and in particular a Single European Railway Area [4]. Most prominently, the EU Shift2Rail Joint Undertaking [5], especially within its Innovation Programme 4, aims to provide users with a "one-stop-shop" solution that allows them to handle multi-modal trips across borders, using a single application that integrates many different services (shopping, booking, ticket issuing, etc.) from heterogeneous providers of different Nations. This requires the integration of services offered by transport operators from different countries, which typically use different standards and specifications to describe data such as travel offers, booking information, etc. This heterogeneity of data representations significantly hinders the interoperability of the systems to be integrated, and it can be mitigated through the adoption of suitable conversion mechanisms between data specifications. Scrocca *et al.* [6] developed a promising data conversion approach, following the schema described in [7] and shown in Figure 1.

In this schema, a reference ontology acts as "pivot" between data specifications A and B, whereby specification A is "lifted" to the ontology (i.e., the concepts in specification A are mapped to those in the ontology), and then the latter is "lowered" to specification B. The approach has proven to be effective [6] and, although it originated from projects focusing on the transportation domain and has been tested using transportation data, it is general and can be applied to any other domain where a reference ontology is available.

At its core, the schema of Fig. 1 relies on declarations—expressed in suitable notations—that precisely establish correspondences between terms of specifications A, B with concepts in the reference ontology. The creation of these mappings, however, is typically a time-consuming activity, which must be carried out by users who have a good level of familiarity with the data specifications and with the reference ontology.

This paper presents a technique that aims at easing the process of creating mappings between concepts in different data specifications and ontologies by: (i) suggesting to users potential mappings between the terms in a specification and those of an ontology; (ii) allowing them to review and confirm/revise the suggested mappings; and (iii) automatically generating the necessary annotations and declarations that enable the conversion technique of [6].

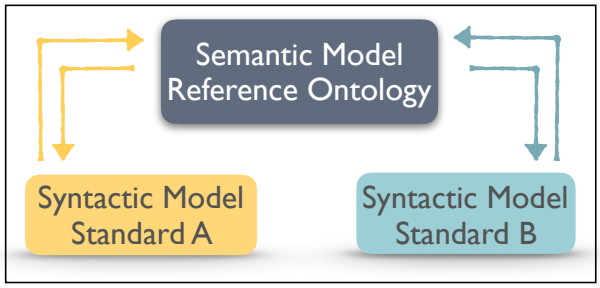

Fig. 1. General schema of conversion mechanism (from [6], [7]).

The technique builds on the principles laid out in [8]. It assumes that data specifications are provided as XSD files, and it is based on two main steps. First, it looks for *linguistic* similarities between the terms of a given data specification and those in the reference ontology, thus creating an initial mapping. Then, it uses the *structure* of the specification and

DOI reference number: 10.18293/SEKE2021-161

of the ontology (i.e., how terms are related to one another) to refine—and possibly extend—the linguistic mapping.

The technique has been implemented in a prototype tool, called SMART (SPRINT Mapping & Annotation Recommendation Tool), as part of the SPRINT [9] project, which aims to define an innovative Interoperability Framework [10]. The tool supports users in the creation and review of the mappings, and then in the generation of the corresponding annotations. The technique and the tool have been validated through a set of mapping experiments involving data specifications and ontologies from the transportation domain.

The paper is structured as follows: Sect. II overviews some relevant related works; Sect. III describes the procedures for generating the suggested mappings and the corresponding annotations; Sect. IV briefly describes the SMART tool; Sect. V presents the results of the validation, and Sect. VI concludes.

II. Related Work

In the domain of mappings between XML-based data and ontologies, most works focus on automatically transforming XML Schemas into newly-created ontologies capturing the implicit semantics existing in the structure of XML documents. For example, Rodrigues *et al.* [11] specify mappings between the elements of an XML Schema and those (classes, object and datatype properties) defined by an OWL ontology. OWL elements are identified by their URI references [12], while the mapped XML nodes are identified by XPath [13] expressions.

When transforming XML-based information into an ontology, two approaches are most common: in the first approach, mapping rules between elements of the XSD and OWL standards are used to generate an ontology from an XSD file; in the second approach, instead, the generated ontology is populated from XML instances. Hacherouf *et al.* [14] focus on the first approach. They use a set of transformation patterns based on the Janus method [15] to translate an XSD block to an equivalent ontology element. The Janus method uses a greater number of XSD elements [16] compared to the work in [11], where transformation rules are limited to the most-used XSD elements (*xsd:element, xsd:attribute, xsd:complexType*). Some works follow a linguistic approach to translate XML-based information into ontologies. Among them, An *et al.* [17] propose a heuristic algorithm for finding semantic mappings based on tree pattern formulas [18]. Yin *et al.* [19] define a method to create mappings between the concepts of two different ontologies. The method divides each ontology into several sub-trees using a classification method [20], and builds mappings between the root nodes of the identified trees in the ontologies. Word similarity is defined based on the assumption that the longer the common substrings between two terms, the more similar they are, and it is computed using the Longest Common Substring algorithm [21]. Shen *et al.* [20] present a method to compute contextual similarity between two words. The idea is that two concepts can be mapped when they either have high word similarity and low context dissimilarity, or low word similarity and high context similarity.

Our proposed technique is unique in that it employs a two-step process that combines both a *linguistic* and a *structural* approach to map elements between XSD specifications and ontologies. This has the advantage that, even when schema elements do not correspond structurally, they might still be linguistically similar, which makes it possible to establish suitable correspondences. Some works (e.g., the Janus method [15]) cover a greater range of XSD features than our approach when transforming XSD schemas into OWL ontologies. However, on the one hand, we pursue a different aim, in that we do not generate new ontologies, but identify correspondences between existing elements; on the other hand, our approach exploits both linguistic and structural features, and we leave for future work the extension of the breadth of XSD features taken into account by the algorithms.

The next section provides some details about the proposed technique.

III. Methodology

The overall workflow of the approach implemented in the SMART tool is depicted in Fig. 2. Given a pair of specifications, SMART identifies a set of mapping suggestions, where each suggestion is a pair of terms—one from each specification—accompanied by a Confidence Score (CS). Then, the selector module receives the mapping suggestions and allows the user to *manually* inspect them; during the inspection, the user can confirm or modify the mappings, and even suggest new pairs, if necessary. Alternatively, the user can let the SMART tool *automatically* choose the suggestions with the highest CS. Finally, the pipeline sends the *Confirmed Mappings* to the *Annotation Generation* module to produce the annotations.

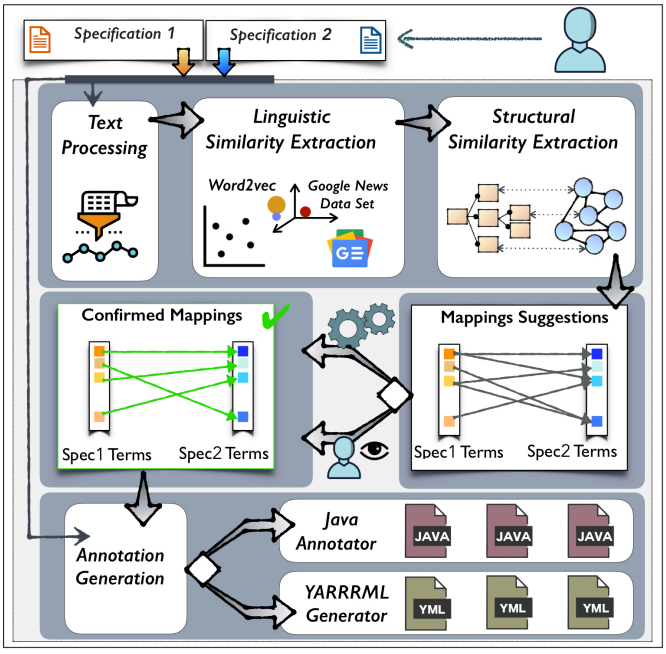

Fig. 2. Overview of the workflow implemented in the SMART tool.

The rest of this section describes the procedure for generating the pairs of suggested mappings, whereas Section IV provides an overview of the implementation of the tool.

A. Mapping Algorithm

Given two different data specifications, the Mapping Algorithm's primary idea is to identify *linguistically* and *structurally* similar terms. The algorithm uses two main techniques: *(i)* linguistic mapping and, then, *(ii)* structural mapping. The former applies Natural Language Processing (NLP) and Machine Learning (ML) techniques to identify similar terms. The latter, instead, exploits the source and target data specifications' structures to refine—and possibly extend—the set of mappings. Algorithm 1 details the flow of the Mapping Algorithm. For clarity and brevity, some parts have been encapsulated in sub-algorithms that are shown as algorithms 2, 3 and 4.

The algorithm takes as input two specifications. Typically, one of them is represented by an XSD file (*X* in line 2 of Alg. 1), whereas the other is an ontology represented by an OWL file (*O*). However, the algorithm can also work when the input files are both XSD files, or both ontologies.

Algorithm 1 Mapping Algorithm

```
1: procedure SMARTMAPPING
2: input: X: XSD file, O: OWL file
3: output P: set of triples ⟨xt, ot, s⟩, xt ∈ X, ot ∈ O, s:Confidence score
4:     xName ← (X.Attribute.name ∪ X.Element.name)
5:     xType ← (X.Attribute.type ∪ X.Element.type)
6:     xCl ← X.ComplexTypes
7:     xObPr ← {xb|xb ∈ xName if XType(xName) = ComplexType}
8:     xDtPr ← {xd|xd ∈ xName if XType(xName) = Datatype}
9:     oCl ← O.Class
10:    oObPr ← O.ObjectProperty
11:    oDtPr ← O.DatatypeProperty
12:
13:    ▷ Create initial mapping between terms using linguistic similarity
14:    mappedClass ← W2VlinguisticMap(xCl, oCl)
15:    mappedObjProp ← W2VlinguisticMap(xObPr, oObPr)
16:    mappedDataProp ← W2VlinguisticMap(xDtPr, oDtPr)
17:
18:    ▷ New mappings between object properties (Alg. 2)
19:    mappedObjProp ← AddObjPropFromClasses(
                           mappedObjProp, mappedClass,
                           xObPr, oObPr )
20:
21:    ▷ New mappings btw. classes based on classes & obj. prop. (Alg. 3)
22:    mappedClass ← AddClassesFromClassesAndObjProp(
                           mappedObjProp, mappedClass,
                           xCl, oCl )
23:
24:    ▷ New mappings between classes based only on properties (Alg. 4)
25:    mappedClass ← AddClassesFromObjProp(
                           mappedObjProp, mappedClass,
                           xCl, oCl )
26:
27:    return mappedClass ∪ mappedObjProp ∪ mappedDataProp
28: end procedure
```

Linguistic Mapping: The proposed linguistic mapping technique exploits the model presented in [22] built using the Word2Vec (W2V) algorithm [23] pre-trained on Google News dataset (about 100 billion words) [24]. In the rest of the work, we refer to this W2V pre-trained model whenever we use the expression *W2V model*. The linguistic mapping technique consists of the following five steps.

Algorithm 2 New Object Properties

```
1: procedure ADDOBJPROPFROMCLASSES
2: input: mappedObjProp, mappedClass, xObPr, oObPr
3: output P: set of triples ⟨xt, ot, s⟩, xt ∈ X, ot ∈ O, s:Confidence score
4:
5:     propList ← ∅
6:     foreach (xc_j, oc_j, s_j) ∈ mappedClass do
7:         foreach (xc_i, oc_i, s_i) ∈ mappedClass do
8:             foreach (xp, op) ∈ xObPr × oObPr do
9:                 if xc_i = xp.ComplexType & oc_i = op.Domain &
10:                    xc_j = xp.Type & oc_j = op.Range then
11:                    s ← (s_i + s_j)/2
12:                    propList ← propList ∪ {(xp, op, s)}
13:                end if
14:            end foreach
15:        end foreach
16:    end foreach
17:    return mappedObjProp ∪ propList
18: end procedure
```

a) Initialization: This step loads the W2V model and outputs its unique terms as a *vocab list* and its *similarity vector* (*i.e.*, the keyed vector representation of the terms).

b) Pre-processing the specifications: We assume that the XSD specification represents the knowledge as a set of ComplexTypes containing Elements and Attributes, along with their types, which can be either DataType or ComplexElement. On the other hand, the ontology represents knowledge as a set of Classes and related Properties. Properties can be either DataTypeProperty or ObjectProperty. A Property corresponds to a relation between its domain (represented by a Class) and its range (which can be a Class or a DataType). Inspired by the mapping rules introduced in [11], we designed a set of transformation rules presented in Table I. Then, this step parses the XSD and the ontology files and builds three sets of terms from each file. Lines 4-11 of Alg. 1 show the steps to obtain these sets. The sets extracted from the XSD file are *xCl*, *xObPr*, *xDtPr*, where *xCl* (resp., *xObPr*, *xDtPr*) is the set of candidate terms to be mapped to Classes (resp., ObjectProperties, DataTypeProperties) in the ontology. Concerning the ontology, instead, *oCl* (resp., *oObPr*, *oDtPr*) is the set of terms corresponding to Classes (resp., ObjectProperties, DataTypeProperties) in the ontology.

In the following steps (which are encapsulated in the application of the W2VlinguisticMap function on lines 14-16), mappings between pairs of terms from the various sets are created, depending on their nature, using a linguistic approach.

c) Finding n *similar terms:* The W2V model is applied separately to the six sets of terms to get n similar words for each term, where n is a configuration parameter (a positive integer). We tested various values for n (3, 5, 10, 20) to find a good balance between accuracy and efficiency of the approach, and we finally settled on $n = 3$. For instance, if x and y are the number of terms from the two specifications (XSD file and ontology), respectively, after obtaining n similar terms using the W2V model, the resulting matrices will have size $x \cdot (n+1)$ (for the XSD file) and $y \cdot (n+1)$ (for the ontology)—notice

TABLE I
XSD TO OWL STRUCTURAL MAPPING RULES

XSD	OWL Type and Name
<xsd:complexType name = **"A"**> <xsd:complexContent> <xsd:extension base = **"B"**> *Where B is another ComplexElement*	Class(**A**) SubClassof(**B**)
<xsd:complexType name = **"A"**> <xsd:complexContent> <xsd:extension> <xsd:element name = **"E1"** type= **"B"**> *Where B is another ComplexElement*	ObjectProperty(**hasE1**) Domain(Class(**A**)) Range(Class(**B**))
<xsd:complexType name = **"A"**> <xsd:complexContent> <xsd:extension> <xsd:attribute name = **"Atr1"** type = **"B"**> *Where B is another ComplexElement*	ObjectProperty(**hasAttr1**) Domain(Class(**A**)) Range(Class(**B**))
<xsd: complexType name = **"A"**> <attribute name = **"Attr1"** type = **"D"**> *Where D is a DataType*	DataTypeProperty(**hasAttr1**) Domain(Class(**A**)) Range(DataType(**D**))
<xsd:complexType name = **"A"**> <xsd:complexContent> <xsd:extension> <xsd:element name = **"E1"** type = **"D"**> *Where D is a DataType*	DataTypeProperty(**hasE1**) Domain(Class(**A**)) Range(DataType(**D**))

that each original term is also included. Table II provides an example of the resulting matrix. The left-most column shows the terms from each specification, whereas the others show the words suggested through the W2V model for each term.

TABLE II
MATRIX REPRESENTATION AFTER STEP (C)

(A): W2V suggestions for terms in the first specification			
Term	**SimilarTerm1**	**SimilarTerm2**	**SimilarTerm3**
Itinerary	Itinerary	Itineraries	CruiseTour
EffectiveDeparture	Departs	SuccessionPlan	DepartsArrives

(B): W2V suggestions for terms in the second specification			
Term	**SimilarTerm1**	**SimilarTerm2**	**SimilarTerm3**
Trip	Trip	Travels	FliesInto
departureTime	departing	abruptdeparture	departures

d) Match terms in the first specification to those in the second specification: In this step, each term from each matrix produced in step (c) for the first specification is matched to the terms in the corresponding matrix for the second specification. Therefore, the matrix obtained for set *xCl* is matched to the one of *oCl* (within the W2VlinguisticMap invocation on line 14), and similarly for the other sets of terms. The W2V similarity vector (discussed in step (a)) is used to compute the cosine similarity [25] (*CS*) for each pair of terms $\langle \text{MatTermS1}, \text{MatTermS2} \rangle$, where MatTermS1 (resp., MatTermS2) is a term of the matrix obtained for the first (resp., second) specification (notice that the matrices include the original terms retrieved from the specifications). *CS* ranges from 0 to 1, where the higher the value, the higher the similarity. The resulting triplets have the form $\langle \text{MatTermS1}, \text{MatTermS2}, CS \rangle$. We set 0.5 as cut-off threshold, and consider a triplet as a potential generator of a mapping (according to the rule described in step (e)) if its *CS* value is greater than or equal to the threshold.

e) Count number of matches: In this step we take each triple $\langle \text{MatTermS1}, \text{MatTermS2}, CS \rangle$ obtained in step (d), and we trace it back to its pair of original terms $\langle \text{TermS1}, \text{TermS2} \rangle$ from the first and second specifications, respectively. For example, consider again Table II. Imagine that, after step (d), we have a triple $\langle \text{Itineraries}, \text{Travels}, 0.677 \rangle$. We trace it back to the original pair of terms $\langle \text{Itinerary}, \text{Trip} \rangle$, and we increase by 1 the counter of the number of matches between Itinerary and Trip. In this way, for each pair of original terms $\langle \text{TermS1}, \text{TermS2} \rangle$ we count the number of matches, and we compute the similarity value $CS_{ts1,ts2}$ for the pair as the average of the *CS* values of the triples $\langle \text{MatTermS1}, \text{MatTermS2}, CS \rangle$ that trace back to it. At the end of this step, we produce a set of triples $\langle \text{TermS1}, \text{TermS2}, CS_{ts1,ts2} \rangle$.

Step (e) concludes the linguistic part of the mapping procedure. The rest of the algorithm (lines 19-27), which is explained next, refines and extends the suggestions by exploiting the structure of the two specifications.

Structural Mapping: The proposed structural mapping technique relies on the rules presented in Table I. More precisely, we use the structure of the ontology as guidance to further refine the mappings returned by the linguistic mapping. First of all, notice that the results produced by the linguistic mapping are stored in three sets of triples of the form $\langle \text{TermS1}, \text{TermS2}, CS_{ts1,ts2} \rangle$ named *mappedClass*, *mappedObjProp* and *mappedDataProp*, where TermS1 and TermS2 are names of Classes, ObjectProperties, or DataProperties, depending on the set. Notice that, for the sake of structural mapping, we consider as Classes, ObjectProperties and DataProperties also elements from XSD specifications when they match the rules of Table I (e.g., a ComplexType is considered, for structural mapping purposes, as a Class).

In this step we consider that each specification defines triples of the form $\langle \text{Domain}, \text{ObjectProperty}, \text{Range} \rangle$. In the following, we indicate a triple from the first (resp., second) specification as $\langle \text{DomainS1}, \text{ObjectPropertyS1}, \text{RangeS1} \rangle$ (resp., $\langle \text{DomainS2}, \text{ObjectPropertyS2}, \text{RangeS2} \rangle$).

We perform the following refinements of the mappings.

(i) Suggest properties if domains and ranges match: If, in *mappedClass*, DomainS1 is mapped to DomainS2 and RangeS1 is mapped to RangeS2, then triple $\langle \text{ObjectPropertyS1}, \text{ObjectPropertyS2}, CS_{ops1,ops2} \rangle$ is added to set *mappedObjProp*, where $CS_{ops1,ops2}$ is the average of the confidence scores of the mappings between domains and ranges—i.e., pair $\langle \text{ObjectPropertyS1}, \text{ObjectPropertyS2} \rangle$ is suggested with confidence score $CS_{ops1,ops2}$. This step is performed by the procedure invoked at line 19 of Alg. 1; in particular, the addition of each single new pair is performed by lines 9-13 of Alg. 2.

(ii) Suggest domains (resp., ranges) if properties and ranges (resp., domains) match: If ObjectPropertyS1 is mapped to ObjectPropertyS2 in *mappedObjProp* and RangeS1 is mapped to RangeS2 in *mappedClass*, then pair $\langle \text{DomainS1}, \text{DomainS2} \rangle$ is suggested with confidence score $CS_{ds1,ds2}$, where $CS_{ds1,ds2}$ is the average of the confidence

scores of the mappings between properties and ranges. Similarly, if ObjectPropertyS1 is mapped to ObjectPropertyS2 and DomainS1 is mapped to DomainS2, then we suggest pair $\langle \text{RangeS1}, \text{RangeS2} \rangle$). This step is performed by the procedure invoked at line 22 of Alg. 1, which is detailed in Alg. 3.

Algorithm 3 New Classes Based on Properties and Classes

```
1: procedure AddClassesFromClassesAndObjProp
2: input: mappedObjProp, mappedClass, xCl, oCl
3: output P: set of triples ⟨xt, ot, s⟩, xt ∈ X, ot ∈ O, s:Confidence score
4:
5:     rangeList, domainList ← ∅
6:     foreach (xc_i, oc_i, s_i) ∈ mappedClass do
7:         foreach (xp_j, op_j, s_j) ∈ mappedObjProp do
8:             foreach (xc, oc) ∈ xCl × oCl do
9:                 ▷ Domain and property are mapped, we map range
10:                if xc_i = xp_j.ComplexType & xc = xp_j.Type &
11:                  oc_i = op_j.Domain & oc = op_j.Range then
12:                    s ← (s_i + s_j)/2
13:                    rangeList ← rangeList ∪ {(xc, oc, s)}
14:                end if
15:                ▷ Property and range are mapped, we map domain
16:                if xc_i = xp_j.Type & xc = xp_j.ComplexType &
17:                  oc_i = op_j.Range & oc = op_j.Domain then
18:                    s ← (s_i + s_j)/2
19:                    domainList ← domainList ∪ {(xc, oc, s)}
20:                end if
21:            end foreach
22:        end foreach
23:    end foreach
24:    return mappedClass ∪ rangeList ∪ domainList
25: end procedure
```

(iii) Suggest domains and ranges if proprieties match: In this case, if ObjectPropertyS1 is mapped to ObjectPropertyS2 in *mappedObjProp*, we suggest pairs $\langle \text{DomainS1}, \text{DomainS2} \rangle$ and $\langle \text{RangeS1}, \text{RangeS2} \rangle$, both with confidence score that is 60% that of the mapping between the properties (i.e., that is equal to $0.6 \cdot CS_{ops1,ops2}$). This step is performed by the procedure invoked at line 25 of Alg. 1 (see also Alg. 4).

Algorithm 4 New Classes Based on Properties

```
1: procedure AddClassesFromObjProp
2: input: mappedObjProp, mappedClass, xCl, oCl
3: output P: set of triples ⟨xt, ot, s⟩, xt ∈ X, ot ∈ O, s:Confidence score
4:
5:     rangeList, domainList ← ∅
6:     foreach (xp, op, s) ∈ mappedObjProp do
7:         foreach (xc_i, oc_i) ∈ xCl × oCl do
8:             foreach (xc_j, oc_j) ∈ xCl × oCl do
9:                 if xc_i = xp.ComplexType & oc_i = op.Domain &
10:                  xc_j = xp.Type & oc_j = op.Range then
11:                    s ← (s * 0.6)
12:                    domainList ← domainList ∪ {(xc_i, oc_i, s)}
13:                    rangeList ← rangeList ∪ {(xc_j, oc_j, s)}
14:                end if
15:            end foreach
16:        end foreach
17:    end foreach
18:    return mappedClass ∪ rangeList ∪ domainList
19: end procedure
```

Although the algorithms presented above assume that one specification is given as an XSD file, and the other as an ontology, indeed they have been adapted to also work when the input specifications have the same format (i.e, they are both XSD files, or both ontologies). More precisely, if both inputs are XSD files, then the algorithm simply performs the same pre-processing step explained in point (b) of the linguistic mapping on both files, to extract a set of "candidate classes" xCl_1, xCl_2 from each file, which are then used in the rest of the algorithm instead of *xCl* and *oCl* (similarly for object and data properties).

B. Annotation Generation

The suggested mappings, following a review of the user, are sent to the the annotation generation module for the final step of the process (see Fig. 2). The annotation generation module is composed of two pipelines, one for each type of annotations supported. More precisely, it can produce either *Java annotations* compatible with the approach presented in [7], or *YARRRML rules* compatible with the converter presented in [6]. Since the annotation generation step is tightly linked to the conversion approach depicted in Fig. 1, it assumes that the first specification is given in terms of an XSD file, whereas the second is an ontology. The rest of this section provides a brief description of the two pipelines.

Java annotation pipeline: In this case, the module analyzes each suggested mapping, and produces a corresponding Java annotation. More precisely, it first determines whether the mapping concerns Classes of Properties. Depending on the case, it fills the appropriate template (i.e., @Rdfs<*TYPE*>("<*ONTOLOGY NAME*>: <*TARGET TERM*>")) and outputs a suitable annotation. For example, lines 2 and 4 in Fig. 3 show a pair of annotations for Class and Property mappings, respectively. For instance, if the term "GeoPoint" (in the XSD specification) is mapped to term "GeoCoordinates", which is a Class in the reference ontology (in this case, the IT2Rail Ontology [26], see also Sect. V), the annotation will be *@RdfsClass("IT2Rail: GeoCoordinates")*. Then, the annotation generation module uses the JAXB package [27] to create, from the XSD file, the Java classes to be annotated. In the final step of the pipeline, the module, for each mapped term of the XSD specification, parses the Java files to find the term's declaration, then it inserts the corresponding annotation appropriately (see lines in bold in Fig. 3). Finally, the user receives a zipped folder containing the annotated Java classes.

YARRRML generation pipeline: YARRRML [28] is a human-readable text-based representation for declarative Linked Data generation rules. The YARRRML generation module first identifies the structural relationships between the terms in the XSD specification. More precisely, the module extracts, for each term of the XSD file corresponding to a Class, its Properties and stores those for which there is a suggestion in the confirmed mappings. Next, according to YARRRML's syntax, the module generates the appropriate *prefixes* and *mappings* blocks. The *prefixes* block contains the required namespaces (e.g., the ontology's namespace). In the *mappings* block, instead, the module defines, for each term corresponding to a Class in the XSD specification, the following three elements: (i) data source location, (ii) subjects' generation, and (iii) predicate-object annotations. As the result, the user receives the YARRRML declarations in a *.yml* file.

```
...
@RdfsClass("IT2Rail:GeoCoordinates")
public class GeoPoint extends FSMID
{   @RdfProperty(propertyName = "IT2Rail:hasLatitude")
    @XmlAttribute(name = "Latitude", required = true)
    protected BigDecimal latitude;
...
```

Fig. 3. Example of Java annotations for Class and a Property (in bold).

TABLE III
DATASETS DESCRIPTION

Specification	Type	Number of terms
NeTEx	XSD	91
FSM	XSD	113
IT2Rail	Ontology	543
Transmodel	Ontology	231
Neptune	XSD	89

IV. TOOL

The current version of the SMART prototype is comprised of two containerized components: a RESTful API and a web server hosting a front-end built with Angular [29], both communicating using JSON standard.

The API is designed to allow multiple simultaneous user requests by relying on the FastAPI [30] framework; the asynchronous environment coupled with the NginX [31] web server allows the tool to manage high throughput of parallel requests. Although NginX can handle up to thousands of requests at once, the API is limited by the number of resources the mapping process requires (e.g., in our experiments, each run was using up to 8GB of RAM): requests that would require SMART to exceed the available processing power are enqueued for when the tool will become available again.

After uploading and selecting the specification files to be mapped, the user can send a mapping request to the server. Every incoming request is given a unique identifier and is handled separately by the tool, allowing it to store and retrieve results asynchronously. Once a request has been parsed, the tool spawns a separate process to handle the computationally heavy mapping process and relinquishes the control until this operation is completed. The process itself can take several minutes to complete.

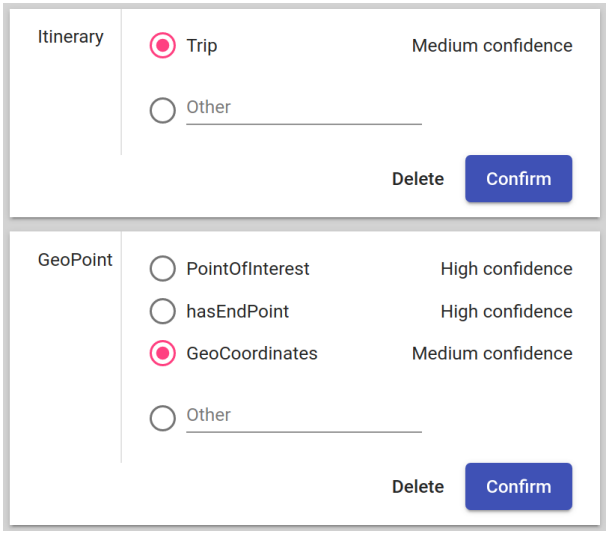

Fig. 4. SMART term selection example.

Available from github.com/alexander-nemirovskiy/s2r_mapping_ui.git.

Once the mapping phase is completed, the user receives a selection of up to 3 most-similar ontology terms for each source term in the XSD file, each associated with its *CS* value (see Sect. III-A) discretized as *high* ($CS \geq 0.75$), *medium* ($0.30 \leq CS < 0.75$) and *low* ($CS < 0.3$) confidence. The user can either confirm the suggested mapping, select one of the alternatives, or add a choice of their own making as they see fit, as shown in Fig. 4. In this case, SMART has selected the ontology term *Trip* as a possible mapping for the XSD term *Itinerary* with a *medium CS* rating. The "Other" input field is used to create a new alternative if the user is not satisfied with SMART recommendations. In addition, Fig. 4 shows that, for the *GeoPoint* term of the XSD file, SMART suggested a term (*PointOfInterest*), but the user decided to change the mapping to *GeoCoordinates*. Alternatively, the user can choose the automatic mapping process, in which the tool automatically selects the term with the highest *CS* value for each pair. After the selection has been made in either way, the user can proceed with the annotation generation phase based on the pipeline selected at the beginning of the mapping process, then the annotated files can be downloaded.

V. VALIDATION

We evaluated the effectiveness of the approach on a few case studies involving specifications from the transportation domain. In particular, this section focuses on the accuracy of the Mapping Algorithm (see Sect. III-A), which is the core of the approach, in terms of its ability to suggest meaningful mappings. More precisely, we took five specifications (XSD files from NeTEx [32], FSM [33] and Neptune [34], and the IT2Rail [26] and Transmodel [35] ontologies) from the transportation domain, we used the Mapping Algorithm to generate suggested mappings between various pairs of specifications, then we manually evaluated the accuracy of the suggestions output by the algorithm (before the review of the user).

Table III collects the basic information (type of specification—XSD file or ontology—and number of terms) about the five specifications used, which are briefly introduced in the following. The *IT2Rail ontology* was created in the project by the same name [26] and it is at the basis of the ontology that is currently being developed within the Shift2Rail Innovation Programme 4. *Transmodel* (short for "Public Transport Reference Data Model") is a European Standard that covers various areas of the transportation

A short video of the tool is available at github.com/alexander-nemirovskiy/s2r_mapping_ui/raw/master/docs/SMART_demo.mp4.

domain, such as network topology representation, scheduling, operation monitoring, fare management, etc. Recently, a full-fledged ontology [35], which we have used for our evaluation, has been defined starting from the Transmodel standard. *NeTEx* [32] is a CEN technical standard for exchanging public transport schedules and data. NeTEx is a large standard, which is divided into three parts, each covering a subset of Transmodel standard. For our experiments, we considered a subset of NeTEx, focusing on the description of the infrastructure (stop points, vehicles, etc.). *FSM* (short for "Full Service Model", [33]) is a standard for representing information about ticketing and reservations in a heterogeneous transport environment. Finally, *Neptune* [34] is the reference format used in France to exchange information concerning public transport (itineraries, timetables, etc.).

Each test case examines a different combination of specifications. Table IV lists the combinations that we tried. For each test case, we carefully assessed the output results to determine the accuracy of the mapping suggestions. For this purpose, we relied on the documentation of each specification describing the terms in the dataset. We categorized each suggested pair as *Correct*, *Incorrect*, *Ambiguous*, or *Unfeasible*.

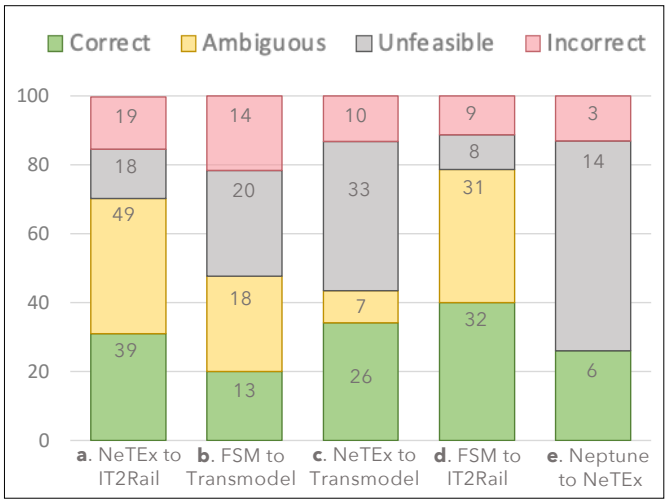

Fig. 5. Detailed results of the evaluation, where the numbers in the bars correspond to the number of pairs in each category, for each test case.

The first two categories are self-explanatory. A pair is deemed *Ambiguous* if there is not enough information about the meaning and usage of a term in the specification to evaluate the correctness of the suggestion. A mapping is considered *Unfeasible* if no equivalent representation of the term in the first specification is available in the second one. Given this categorization, the accuracy of the results of each test case was computed as the percentage of the *Correct* mappings over the sum of *Correct* and *Incorrect* ones. Both the *Ambiguous* and the *Unfeasible* categories are excluded from the computation as they do not provide a meaningful contribution for it, since the pair either lacks a clear definition, or there is no alternative for the first term in the second specification.

TABLE IV
VALIDATION RESULTS (WHERE *Ambiguous* AND *Unfeasible* MAPPINGS ARE NOT CONSIDERED WHEN COMPUTING THE ACCURACY)

Spec1	Spec2	Accuracy	Execution Time
NeTEx	IT2Rail	67%	360s
FSM	Transmodel	48%	312s
NeTEx	Transmodel	72%	240s
FSM	IT2Rail	78%	360s
Neptune	NeTEx	67%	345s

Table IV presents, for each test case, the corresponding accuracy, and the time that it took to generate the suggested mappings using the procedure of Sect. III-A. For our experiments, we deployed the tool on a general-purpose Amazon EC2 instance with 32GB Memory, 8 vCPU 3.0 GHz Intel Xeon processor, and up to 1Gb/s connection speed. The duration of the mapping generation process depends on the size of the input specifications; however, as shown in Table IV, no experiment took more than 6 minutes. For completeness' sake, Figure 5 provides, in addition to a graphical representation of the share of each category of mapping for each test case, the number of elements in each category. On average, SMART's accuracy is 66.4%, ranging from 48% to 78% (recall that, when computing the accuracy, we do not count *Ambiguous* and *Unfeasible* mappings in the denominator of the ratio).

VI. CONCLUSION

This paper presented a tool-supported approach to suggest mappings between terms of separate data specifications; this is the basis for converting data between different data formats, which is a core enabler of interoperability in heterogeneous Systems of Systems. The approach, which is also able to automatically create, from selected mappings, annotations compatible with the conversion mechanisms introduced in [6], [7], has been validated on various test cases from the transportation domain, showing promising results.

In the future, we will refine both the underlying suggestion mechanism and the supporting SMART tool. In particular, we plan to explore the possibility of using domain-specific models (e.g., tailored to the transportation domain) for the linguistic mapping part of the procedure, instead of the general-purpose one used in this work; this would improve the accuracy of the first step of the procedure. In addition, we plan to extend the structural mapping part of the algorithm with richer rules, able to handle a wider range of features of XSD files.

ACKNOWLEDGMENT

This work was supported by Shift2Rail and the EU Horizon 2020 research and innovation programme under grant agreement No: 826172 (SPRINT).

REFERENCES

[1] M. Jamshidi, *Systems of systems engineering: principles and applications*. CRC press, 2017.
[2] P. Uday and K. Marais, "Designing resilient systems-of-systems: A survey of metrics, methods, and challenges," *Systems Engineering*, vol. 18, no. 5, pp. 491–510, 2015.

[3] "Roadmap to a single european transport area – towards a competitive and resource efficient transport system," European Commission, White Paper. [Online]. Available: https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2011:0144:FIN:EN:PDF

[4] "Single european railway area: better rules on interoperability, noise and access for persons with reduced mobility." [Online]. Available: https://ec.europa.eu/transport/modes/rail/news/2019-05-16-single-european-railway-area_en

[5] "Shift2Rail." [Online]. Available: http://www.shift2rail.org/

[6] M. Scrocca, M. Comerio, A. Carenini, and I. Celino, "Turning transport data to comply with EU standards while enabling a multimodal transport knowledge graph," in *The Semantic Web – ISWC 2020*, ser. LNCS, 2020, pp. 411–429.

[7] A. Carenini, U. Dell'Arciprete, S. Gogos, M. M. Pourhashem Kallehbasti, M. G. Rossi, and R. Santoro, "ST4RT–semantic transformations for rail transportation," in *Transport Research Arena*, 2018, pp. 1–10.

[8] M. Hosseini, S. Kalwar, M. G. Rossi, and M. Sadeghi, "Automated mapping for semantic-based conversion of transportation data formats," in *Int. Work. On Semantics For Transport*, vol. 2447, 2019, pp. 1–6.

[9] "SPRINT – Semantics for PerfoRmant and scalable INteroperability of multimodal Transport." [Online]. Available: http://sprint-transport.eu

[10] M. Sadeghi, P. Buchníček, A. Carenini, O. Corcho, S. Gogos, M. Rossi, R. Santoro *et al.*, "Sprint: Semantics for performant and scalable interoperability of multimodal transport," in *8th Transport Research Arena TRA 2020*, 2020, pp. 1–10.

[11] T. Rodrigues, P. Rosa, and J. Cardoso, "Mapping xml to exiting owl ontologies," in *Int. Conf. WWW/Internet*. Citeseer, 2006, pp. 72–77.

[12] "owluri." [Online]. Available: http://www.w3.org/Addressing/

[13] J. Clark, S. DeRose *et al.*, "XML path language (xpath)," 1999.

[14] M. Hacherouf, S. Nait-Bahloul, and C. Cruz, "Transforming XML schemas into OWL ontologies using formal concept analysis," *Software & Systems Modeling*, vol. 18, no. 3, pp. 2093–2110, 2019.

[15] I. Bedini, C. Matheus, P. F. Patel-Schneider, A. Boran, and B. Nguyen, "Transforming xml schema to owl using patterns," in *IEEE 5th Int. Conference on Semantic Computing*. IEEE, 2011, pp. 102–109.

[16] M. Hacherouf, S. Nait-Bahloul, and C. Cruz, "Transforming XML documents to OWL ontologies: A survey," *Journal of Information Science*, vol. 41, no. 2, pp. 242–259, 2015.

[17] Y. An, A. Borgida, and J. Mylopoulos, "Constructing complex semantic mappings between XML data and ontologies," in *International Semantic Web Conference*. Springer, 2005, pp. 6–20.

[18] A. Marcelo and L. Leonid, "XML data exchange: Consistency and query answering," in *Proc. of the Symp. on Princ. of Database Systems*, 2005.

[19] C. Yin, J. Gu, and Z. Hou, "An ontology mapping approach based on classification with word and context similarity," in *12th Int. Conference on Semantics, Knowledge and Grids (SKG)*. IEEE, 2016, pp. 69–75.

[20] Z. Zhen, J. Shen, and S. Lu, "Wcons: An ontology mapping approach based on word and context similarity," in *Int. Conf. on Web Intelligence and Intelligent Agent Technology*, vol. 3, 2008, pp. 334–338.

[21] M. A. Babenko and T. A. Starikovskaya, "Computing the longest common substring with one mismatch," *Problems of Information Transmission*, vol. 47, no. 1, pp. 28–33, 2011.

[22] "word2vec tool." [Online]. Available: https://code.google.com/archive/p/word2vec/

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[24] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *arXiv preprint arXiv:1310.4546*, 2013.

[25] D. Jatnika, M. A. Bijaksana, and A. A. Suryani, "Word2vec model analysis for semantic similarities in english words," *Procedia Computer Science*, vol. 157, pp. 160–167, 2019.

[26] "IT2Rail – Information Technologies for Shift2Rail." [Online]. Available: http://it2rail.eu/

[27] J. Fialli and S. Vajjhala, "The java architecture for XML binding (JAXB)," *JSR Specification, January*, 2003.

[28] P. Heyvaert, B. De Meester, A. Dimou, and R. Verborgh, "Declarative Rules for Linked Data Generation at your Fingertips!" in *Proceedings of the 15th ESWC: Posters and Demos*, 2018.

[29] Google, "Angular." [Online]. Available: https://angular.io/

[30] S. Ramírez, "FastAPI." [Online]. Available: https://fastapi.tiangolo.com/

[31] F5, Inc., "Nginx." [Online]. Available: https://www.nginx.com/

[32] "NeTEx." [Online]. Available: https://github.com/NeTEx-CEN/NeTEx/tree/master/xsd

[33] "FSM – Full Service Model." [Online]. Available: https://tsga.eu/fsm

[34] "Neptune." [Online]. Available: http://www.normes-donnees-tc.org/category/neptune/

[35] "transmodel." [Online]. Available: https://oeg-upm.github.io/snap-docs/tm-organisations.owl/documentation/index-en.html