Alignment Format for Semantic Translation

Paweł Szmeja, Wiesław Pawłowski, Maria Ganzha, Marcin Paprzycki, and Katarzyna Wasielewska-Michniewska

Abstract Abundance of vastly heterogeneous, high-volume/high-velocity data producers/consumers, predominantly caused by proliferation of IoT-based solutions, results in an urgent need for efficient semantic interoperability solutions. Hence, the need to solve the problems of domain understanding, domain formal representation, and expression of mappings between different data models arise. In this contribution, we present an alignment format called IPSM-AF, for persisting correspondences between ontologies that can be used for semantic translation. Specifically, alignments represented using the IPSM-AF can be efficiently parsed and consumed by the IPSM tool, which performs the actual translation (possibly on streaming data). The proposed format is compliant with the Alignment API format, level 2.

Keywords Semantic translation · Semantic stream processing · Ontology · Alignments · Internet of Things

1 Introduction

The rise of the Internet of Things [2, 18], in which "everything" can be connected, and used as data publisher or subscriber, leads to new challenges in data processing. IoT-based solutions either already are or very soon will be one of the foundational ingredients of almost every modern technological solution. Concepts such as Smart City, Smart Grid, or Supervisory Control and Data Acquisition (SCADA) systems are just a few, but prominent examples. With rapidly growing number of deployed IoT artifacts (platforms, applications, devices), one of the key problems is the ability to understand, integrate, and uniformly process data, materializing/existing in

P. Szmeja · M. Ganzha · M. Paprzycki · K. Wasielewska-Michniewska
Systems Research Institute, Polish Academy of Sciences, Newelska 6, 01-447 Warsaw, Poland

W. Pawłowski (✉)
Faculty of Mathematics Physics and Informatics, University of Gdańsk, Wita Stwosza 57, 80-308 Gdańsk, Poland
e-mail: wieslaw.pawlowski@ug.edu.pl

V. Mahajan et al. (eds.), *Sustainable Technology and Advanced Computing in Electrical Engineering*, Lecture Notes in Electrical Engineering 939,
https://doi.org/10.1007/978-981-19-4364-5_47

different formats and semantics, often in heterogeneous domains. The task of data format conversion/translation, also called *syntactic translation*, is reasonably well understood, and there are many tools, both open-source and commercial, that support it. Data interoperability/integration at the level of semantics, on the other hand, poses a much bigger challenge, especially in IoT environments where, naturally occurring, high volume/velocity, heterogeneous data *streams* have to be considered and efficiently handled. The viability and the need for integration of heterogeneous data streams in IoT, including real-time analytics, is well demonstrated, understood, and often includes semantic data [14]. The solutions are, however, limited to systems that already implement common semantics and operate on directly interoperable and exchangeable data. Communication between semantically incompatible systems requires the addition of semantic translation.

The problem of "meaningful communication" (data exchange) between IoT artifacts becomes even more pronounced over time. Typically, initial deployments of IoT ecosystems exist within homogeneous, domain- or vendor-specific "silos," which do not need to communicate externally. For instance, companies *X* and *Y* installed smog monitoring ecosystem, while companies *A*, *B*, and *C* provided (in the same city) infrastructure for electric scooters rental (with GPS sensors build into them, to be able to track their location). Each company used its own data representation and semantics to deal with geospatial data. Now, the City would like to combine information from both ecosystems to develop a novel user-centric application, where smog data is combined with scooter location to help user decide, which road to take. Here, for all practical purposes, it cannot be mandated, that either the providers of the smog alert data, or the electric scooter rental companies, are required to reengineer their systems to adapt the geospatial data model to the one used by the other (or to yet another data model that the City is supporting). Situations like this are becoming more and more common, as numerous IoT-based solutions/services are being combined to provide users with ubiquitous "smart environments."

In our earlier works [8, 9], we have proposed a viable solution to the semantic interoperability problem, based on the concept of a *semantic mediator*, offering (streaming) *semantic translation* capabilities for data. The mediator takes "translation rule-sets" as arguments, and by "applying" them, creates appropriate *translation channels*. An efficient and scalable [12] implementation was subsequently created within the INTER-IoT project [7, 17],[1] and later further enhanced and adopted as the "kernel part" of the interoperability solution for the ASSIST-IoT project.[2] The architecture of the *Inter-Platform Semantic Mediator* (IPSM) utilizes a concept of *modular central ontology*, to minimize the number of necessary translation rule-sets. Since the data models of artifacts are assumed to be formally represented in the form of an ontology,[3] it is natural to consider (partial) *alignments* between these ontologies, and the central ontology, as the basis for the rule-sets. "Partiality" of the

[1] http://www.inter-iot-project.eu.

[2] https://assist-iot.eu/.

[3] Comprehensive discussion concerning what needs to be done if the original local data representation is not ontology-based can be found in [5, 11].

alignments reflects the fact that *only* the information that is to be communicated/exchanged between pertinent artifacts needs to be taken into account. This, in turn, is a consequence of the assumption that all details of data streams that are to be translated, are fully known to the IoT ecosystems developers. As a consequence, the alignments required for the translation do not have to cover complete data models of communicating artifacts and therefore have reasonable both size and complexity. As a result, translation time is limited, as it involves only information needed in the context of a given "type of conversation." Here, note that this assumption, which is grounded in real-world scenarios observed during the INTER-IoT project (and the remaining six EU projects from the same call) makes an important difference in what follows. While there exists a body of research, which includes, also, an ontology matching competition,[4] it focuses on establishing relationships between complete data models. While this academic problem is very challenging and stimulating, it is not necessarily required to facilitate semantic interoperability between IoT deployments, where data streams are produced, exchanged, and consumed.

Whereas in ontology matching competitions, the main use of an alignment format is to enable direct comparison between outputs of different automatic or semiautomatic tools [16], it is not required to be human-readable, highly expressive, and "executable" as input for data transformation. In other words, the requirements that it answers do not fully cover the challenges of dynamic data translation in IoT, which, for us, was the starting point.

The focus of this paper, and its main contribution, is to discuss an alignment format introduced for formally representing and persisting mappings between different semantic domain models that are consumed by the IPSM tool, to perform semantic translation. Since the initial version, described in [21], both the alignment format and the IPSM tool have been updated and extended in different ways. In particular, support for an elegant, concise, and much more readable/comprehensible syntax, based on the RDF Turtle was added. As will be shown, while there exists a large body of work devoted to ontology alignments and their representation, they seem to be slightly off the mark, when application of semantic technologies to *streaming translation* is concerned. The aim of this work is not only to propose the format but also to show how the IPSM-AF can be used in real-world applications. Examples will be based on actual pilot implementations that have been delivered during the INTER-IoT project. To this effect, we proceed as follows. First, we will discuss existing approaches for persisting alignments, then we will introduce the alignment format that we propose, and finally, we will conclude by giving examples of its application.

[4] http://oaei.ontologymatching.org/.

2 Quest for the Mapping Language

Even though ontologies are said to provide a "formal specification of a shared conceptualization" [1], in real-life, models of concepts represented in ontologies can differ in many ways. A domain can be perceived from different perspectives, depending on the modeling purpose or/and approach. Starting from the most simple cases—the same term can describe different concepts, or different terms can be used to denote the same concept. The simplest example can be expressing the same term in different languages, e.g., *name* and *Vorname*, using abbreviations, e.g., *temperature* and *temp*, or using synonyms, e.g., *car* and *automobile*. More involved differences can result from the adoption of different modeling paradigms and/or granularity levels, e.g., address with all components contained in a text literal, or address decomposed into attributes like country, city, street, house and flat number, or decision what should be represented by a concept and what by a property. Of course, the list of examples continues and is easy to construct.

The process of discovering relations/correspondences between entities originating from a pair of ontologies, such as classes, properties, or individuals, is called *aligning* or *matching*. Because of the possible modeling differences between ontologies, the matching task can be (and in most cases is) highly non-trivial, even though some guidelines and patterns have been identified and described in the literature [19, 20]. The result of the matching process is called an *alignment*. At a very abstract level, an alignment can be seen as a set of correspondences, i.e., triples (e_A, r, e_B), where e_A and e_B are entities from ontologies A and B, respectively (or expressions that use said entities), and r is a relation, such as *subsumption* and *equivalence* [4, 24]. Alignments have been utilized as the basis for many ontology-related operations such as ontology merging, query rewriting, or data integration.

In order to "apply" the correspondences forming the alignment to any task, however, they have to be represented/persisted using a *mapping language*. Obviously, a particular choice of the mapping language highly depends on the intended application scenario, but also on the complexity and the level of sophistication of the correspondences constituting the alignment. The more involved the alignment correspondences are, the more *expressive* mapping language they require.

Within the INTER-IoT project, we were interested in utilizing alignments in the process of *semantic translation* of (high velocity) *streams* of IoT-originated data. Here, let us note that, to the best of our knowledge, alignments have not been directly applied in such a scenario before. Therefore, we have started our quest for the "right" mapping language by considering the available options. In [10], we have analyzed existing tools for ontology matching. Here, one of their most important features was the mapping languages they used to persist the matching results (as these matchings were to be used in the dynamic translation process). We have found that, although in some cases alignments were encoded in different application-specific formats, the standard way to express mapping(s) between (complete) ontologies is to use the Alignment API format [3].

Apart from the Alignment API, there were, of course, also other possibilities to consider. The technologies relevant to our needs were not limited to those, that directly express alignments, but also included those, that are capable of expressing mappings between any ontologically described data, or even one-way transformations, that could potentially be used to build alignments. Consequently, the research included alignment, mapping or transformation languages and technologies for OWL or RDF, such as SPARQL,[5] its extension SPARQL Inference Notation (SPIN),[6] Semantic Web Rule Language (SWRL),[7] ShEx,[8] SHACL,[9] SILK [25], RDF Mapping Language (RML[10]) and RMLEditor [13], Simple Knowledge Organization System (SKOS)[11] [15], and even the Web Ontology Language OWL[12] itself. However, there were three important aspects/features of the needed mapping language that we were particularly interested in. We wanted it to: (1) be expressive, in particular allowing flexible *value processing/transformation*; (2) provide clean and *comprehensible* syntax for expressing alignments; (3) have tools that would support it in our scenario (translation of streaming data). Upon in-depth analysis, we have established that none of, just listed, "secondary" options met all three of these requirements at once. Therefore, they have been excluded from further considerations.

The Alignment API format that we identified as the most widely used (and possibly useful for our purpose) defines several *levels*, where higher level indicates higher expressiveness. Specifically, the available levels are: 0—not depending on any particular language correspondences between discrete entities (identified by URIs), 1—replaces pairs of entities by pairs of sets (or lists) of entities, 2—directional, and language-dependent, sets of expressions of a particular language with variables in these expressions.

Unfortunately, as we have established in our work, and reported in [10], even if a tool produced Alignment API compliant mapping, it was on level 1 (only simple correspondences without transformations). Moreover, even then, we did not find any tool that could consume such alignment file and perform automatic translation based on the defined mappings.

On the other hand, there is a detailed description language for expressing alignments on the Alignment API format level 2. It is the EDOAL (Expressive and Declarative Ontology Alignment Language[13] [3, 4]). In principle, it allows to represent complex correspondences, to precisely describe the relation between the entities including: construction, restrictions, transformations, and conditions for equivalence. However, from the point of view of our intended application, i.e., streaming semantic

[5] https://www.w3.org/TR/sparql11-overview/.

[6] https://www.w3.org/Submission/spin-overview/

[7] https://www.w3.org/Submission/SWRL/

[8] https://shex.io/

[9] https://www.w3.org/TR/shacl/.

[10] https://rml.io/

[11] https://www.w3.org/TR/skos-reference/

[12] https://www.w3.org/TR/owl2-overview/.

[13] http://alignapi.gforge.inria.fr/edoal.html.

translation, it did not offer enough support for the "transformations," which according to the official documentation are "still work in progress." Moreover, although EDOAL allows to state *directionality* of individual mappings, it does not express it explicitly in the alignment's header, which seems important and natural in the context of (streaming) translation. Our investigation also showed that there were no alignment creation tools that would actually use the full expressive power of EDOAL format for their output.[14] Therefore, we had to accept the fact, that alignments will need to be mostly "handcrafted." Moreover, in most cases, we could not assume that they will be created by experienced ontology engineers. Therefore, we needed a mapping language, which would seem familiar and understandable to a "working software engineer." As its name suggests, EDOAL mainly (although not exclusively) supports the *declarative* style of defining mappings. This style, although elegant and often more concise, tends to be considerably more difficult to understand and use by practitioners than the *imperative/procedural* one. This aspect became also an important factor in our quest.

Consequently, we have decided to design a dedicated mapping language that will be sufficiently expressive and will have comprehensible syntax dedicated to semantic translation. The proposed solution, called IPSM Alignment Format (IPSM-AF), is compliant with the Alignment API format, allows to express complex correspondences involving, e.g., transformations, supports the imperative/procedural style of defining alignments, and is directly supported by the Inter-Platform Semantic Mediator (IPSM) translation tool. Among advantages of being compliant with the Alignment API are the possibility to store alignments in the Alignment Server,[15] which enables different actors to share available alignments, networks of ontologies, and ontology matching methods.

3 IPSM Alignment Format

Although semantically annotated data can be, in principle, expressed in many different ways, today, the unquestionable standard is the Resource Description Framework (RDF).[16] Therefore, we have decided that the alignment format should follow the assumption that data is expressed in the form of RDF graphs, whereas IPSM-AF allows to persist translation rules between (selected fragments of) RDF graphs that express the pair of underlying ontologies. For the format to be flexible and expressive, IPSM-AF was defined as a level 2 Alignment API format extension. Therefore, it is language-dependent and allows specifying compound constructs. In particular, while designing it we have assumed that it should:

[14] The situation has not changed much, ever since, as recently confirmed in [23, 24].

[15] http://alignapi.gforge.inria.fr/server.html.

[16] https://www.w3.org/RDF/.

- be able to express mappings of sets of expressions, e.g., RDF graphs (not only individual triples or entities),
- allow using *variables* for expressing RDF graph patterns,
- utilize callable *functions*, to perform value transformations, e.g., concatenate strings, apply regular expression to select part of the value, perform arithmetic operations, etc.
- enable expressing restrictions on datatypes, both in source and in target patterns,
- be able to remove parts of the source RDF graph and to add new elements to it.

It should be stressed that these assumptions were based, in large part, on the requirement analysis of the use cases of the INTER-IoT project. Therefore, they are grounded in real-world needs of streaming semantic translations, in IoT ecosystems.

To introduce the proposed format, let us start by considering the overall structure of an IPSM-AF alignment, expressed in RDF/XML as shown in Listing 1. The main element—an *align:Alignment*—contains three "sections," giving the alignment *metadata*, specifying the *steps*, i.e., the ordering, in which individual mappings should be considered, and defining the mappings themselves. The namespace prefixes that are used in the examples that follow are expanded in Table 1.

```
<rdf:RDF xmlns="http://www.inter-iot.eu/sripas#"% other xml namespaces % >
    <align:Alignment>
        %--------------------%
        % alignment metadata %
        %--------------------%
        %--------------------%
        % alignment "steps" %
        %--------------------%
        %--------------------%
        % alignment mappings %
        %--------------------%
    </align:Alignment>
</rdf:RDF>
```

Listing 1 IPSM-AF - alignment structure

The *metadata* section, as illustrated in Listing 2, provides the *title* and *version*, specifies *creator* and gives a short *description* of the alignment. These items, in particular the name, version, and creator, allow for tracking changes and using versioning schemes, which are necessary procedures in any evolving ecosystem. Since the IPSM-AF is compliant with the Alignment API, the metadata section also needs to specify the needed Alignment-API-specific properties. In particular, the *align:level* whose value "2IPSM," states that the alignment is on "level 2" in the Alignment API terminology. The elements *align:onto1* and *align:onto2*, structure of which is also defined by the Alignment API Format, specify the *source* and *target* ontologies, for the alignment, respectively. The IPSM-AF alignments are always considered to be unidirectional—mapping entities of the ontology *align:onto1* to entities of the ontology *align:onto2*. Here, the source ontology can be that of the IoT artifact and the target ontology can be a central ontology, used in the ecosystem. This assumption allows the creator to focus on the translation-targeted task, without the need to consider "reversibility" of the individual mappings defined by the alignment. If translation is needed in both directions then two alignments should be created. The

last metadata item, presented in Listing 2, is the element *sripas:cellFormat*, which specifies syntax used for representing RDF graph patterns, contained in the alignment mappings. By default, it is RDF/XML but IPSM also supports Turtle, which is more concise and readable.

```
<dcelem:title> % alignment title % </dcelem:title>
<exmo:version> % alignment version % </exmo:version>
<dcelem:creator> % alignment creator % </dcelem:creator>
<dcelem:description> % alignemnt description % </dcelem:description>

<align:level>2IPSM</align:level>
% other Alignment API specific metadata %

<align:onto1> % source ontology specification % </align:onto1>
<align:onto2> % target ontology specification % </align:onto2>

<sripas:cellFormat>
    <iiot:DataFormat rdf:about="&sripas;turtle"/>
</sripas:cellFormat>
```

Listing 2 IPSM-AF alignment metadata

The alignment *steps* section (see, Listing 3) allows to specify the *default* order, in which specific mappings of the alignment should be considered ("applied"), possibly allowing any of them to be used more than once or, perhaps, excluded from the translation (i.e., not used at all). When defining a specific *translation channel*, based on the alignment, the IPSM tool allows to overwrite the default sequence of steps, if needed. Each step (represented by the *sripas:step* element) refers to a specific mapping within the alignment via its *identifier* represented by URI. The idea of steps reflects the *procedural* style of IPSM-AF that makes the format more accessible for the less "semantically proficient" users.

```
<sripas:steps rdf:parseType="Literal">
    <sripas:step sripas:order="1" sripas:cell="cell_id"/>
    % more alignment steps %
</sripas:steps>
```

Listing 3 IPSM-AF - alignament steps specification

```
<align:map>
    <align:Cell rdf:about="&sripas;cell_id">
        <align:entity1 rdf:datatype="&xsd;string">
            % source RDF graph pattern %
        </align:entity1>
        <align:entity2 rdf:datatype="&xsd;string">
            % target RDF graph pattern %
        </align:entity2>
        <align:relation>=</align:relation>
        % optional transformations, filters and typing restrictions %
    </align:Cell>
</align:map>
```

Listing 4 IPSM-AF - alignment map/cell

The mappings section of the alignment consists of a sequence of *align:map* elements, each containing a single *cell*, defining a specific correspondence between two *entities* (or compound entity descriptions). Entity can be a class, instance/individual,

Table 1 Namespaces in IPSM-AF

Prefix	Namespace
sripas	http://www.inter-iot.eu/sripas#
var	http://www.inter-iot.eu/sripas:node_
pred	http://www.inter-iot.eu/sripas:pred_
align	http://knowledgeweb.semanticweb.org/heterogeneity/alignment#
dcelem	http://purl.org/dc/elements/1.1/
exmo	http://exmo.inrialpes.fr/align/ext/1.0/#
xsd	http://www.w3.org/2001/XMLSchema#

object, or datatype property. Cells are elements, in which all the mapping logic of the alignment is expressed. The structure of a cell has been outlined in Listing 4. Within *align:entity1*, the source RDF graph pattern is specified that is matched against input ontology. The *align:entity2* defines the target structure of the result of the mapping. When the alignment cell is used/applied, by a translator tool, then any RDF content matching the pattern in the *align:entity1*, will be modified into an RDF graph matching the pattern from *align:entity2*. To relate parts of the source and the target RDF graph patterns, IPSM-AF utilizes *variables* appearing in the *entity1* that can be later referenced from the *entity2* and, possibly, also used by the optional cell components, described in the next paragraph. Variables should belong to the namespace *var*. For instance, *var:A* is expanded to http://www.inter-iot.eu/sripas:node_A. Variables can be used in places of data and object properties' values. They serve as placeholders for values/entities that match the pattern. Additionally, each mapping cell can have optional elements: *sripas:transformation*, *sripas:filters*, and *sripas:typings*. These allow to, respectively, perform function calls during cell application, filter entities based on datatypes, and assign datatypes to entities in the target RDF graph. Listing 5 shows the structure that is to be used to include optional elements. If, for example, in the transformation, a reference to a SPARQL function is passed, the tool that consumes IPSM-AF content should be able to use any RDF library that supports SPARQL to apply the function to arguments specified by the *sripas:param* elements. For instance, the IPSM internally utilizes Apache Jena, which makes it possible to define and employ arbitrary external function libraries, thanks to the ARQ query engine.

```
<sripas:transformation rdf:parseType="Literal">
    <sripas:function about="{function name}">
        <sripas:param order="1" val="{simple value}"/>
        <sripas:param order="2" about="{variable URI}"/>
        % more parameter descriptions %
        <sripas:return about="{variable URI}"/>
    </sripas:function>
    % more function application descriptions %
</sripas:transformation>
<sripas:filters rdf:parseType="Literal">
    <sripas:filter about="{variable URI}" datatype="{datatype}"/>
    % more filters %
```

```
    </sripas:filters>
    <sripas:typings rdf:parseType="Literal">
        <sripas:typing about="{variable URI}" datatype="{datatype}"/>
        % more typings %
    </sripas:typings>
```

Listing 5 IPSM-AF - optional cell properties

4 Basic Translation Constructs

Thus far we have introduced the general structure of alignments, expressed in IPSM-AF, and indicated that the "translation cells" are the "workhorse" of the semantic translation. Let us now discuss some basic constructs that can be used within such translation cells. These constructs can be further composed, to define more complex patterns and correspondences, to be matched against the input data (RDF graphs). Since every RDF graph is a set of *triples*, "atomic" translations that can be applied to such graphs work on triples. Let us start with the case of translating just the *predicate* connecting the subject and the object of a triple. Listing 6 shows an example, in which the predicate of all matching triples will be changed from *ont1:hasName* to *ont2:hasVorname*.

```
    <align:Cell rdf:about="&sripas;name_mapping_example">
        <align:entity1 rdf:datatype="&xsd;string">
            var:X ont1:hasName var:Y .
        </align:entity1>
        <align:entity2 rdf:datatype="&xsd;string">
            var:X ont2:hasVorname var:Y .
        </align:entity2>
        <align:relation>=</align:relation>
    </align:Cell>
```

Listing 6 IPSM-AF triple mapping

Let us now assume that our goal is to "translate" a specific URI. In any alignment cell both *entity1* and *entity2* are RDF graph patterns, i.e., sets of RDF triple patterns. Therefore, to change the URI we need to explicitly use it in the *entity1* triple pattern, and substitute it with the "replacement" URI in the *entity2* pattern. Listing 7 presents a cell defining such a translation. In this example, *ont1:res1* is equivalent to *ont2:res1* in the target ontology.

```
    <align:Cell rdf:about="&sripas;uri_mapping_example">
        <align:entity1 rdf:datatype="&xsd;string">
            var:X a sosa:Observation ;
            sosa:hasResult ont1:res1 .
        </align:entity1>
        <align:entity2 rdf:datatype="&xsd;string">
            var:X a sosa:Observation ;
            sosa:hasResult ont2:res1 .
        </align:entity2>
        <align:relation>=</align:relation>
    </align:Cell>
```

Listing 7 IPSM-AF URI mapping

Listing 8 shows an example of a mapping where, in the target semantics, we want to specify new named entity, generated based on an entity from the source ontology, matched by a variable. To create the named entity, a standard SPARQL IRI function is used.

```
<align:Cell rdf:about="&sripas;named_entity_example">
    <align:entity1 rdf:datatype="&xsd;string">
        var:elem a pt:Element ;
        pt:hasDeviceId var:device_id .
    </align:entity1>
    <align:entity2 rdf:datatype="&xsd;string">
        var:device a iiot:IoTDevice, sosa:Sensor ;
        iiotex:hasLocalId var:device_id .
    </align:entity2>
    <align:relation>=</align:relation>
    <sripas:transformation rdf:parseType="Literal">
        <function about="IRI">
            <param order="1" about="&var;device_id"/>
            <return about="&var;device"/>
        </function>
    </sripas:transformation>
</align:Cell>
```

Listing 8 IPSM-AF named entity

When creating an alignment, it is often necessary to remove triples matching a given triple pattern. An example of a cell defining such a "translation" is depicted in Listing 9. The "removal" is achieved by mapping the source triple pattern to an empty pattern.

```
<align:Cell rdf:about="&sripas;triple_removal_example">
    <align:entity1 rdf:datatype="&xsd;string">
        var:X ont1:hasName var:Y .
    </align:entity1>
    <align:entity2 rdf:datatype="&xsd;string">
    </align:entity2>
    <align:relation>=</align:relation>
</align:Cell>
```

Listing 9 IPSM-AF remove part of a graph

Finally, Listing 10 shows an example of adding new triples to an RDF graph. The original triple is preserved, but two additional are included. Here, we assume that the values of variables *var:V* and *var:Z* can be calculated from the result of applying the cell transformation (otherwise, the cell would be ill-formed).

```
<align:Cell rdf:about="&sripas;adding_new_triples">
    <align:entity1 rdf:datatype="&xsd;string">
        var:X ont1:hasName var:Y .
    </align:entity1>
    <align:entity2 rdf:datatype="&xsd;string">
        var:X ont1:hasName var:Y .
        var:X ont1:hasFirstName var:V .
        var:X ont1:hasName var:Z .
    </align:entity2>
    <align:relation>=</align:relation>
    <sripas:transformation rdf:parseType="Literal">
        % transformation %
    </sripas:transformation>
</align:Cell>
```

Listing 10 IPSM-AF add triples to a graph

It is important to note, that application of the alignment cell translates all the triples of an RDF graph that match the *align:entity1* graph pattern, preserving at the same time all the unmatched ones.

Based on analysis of the literature, requirements analysis of the use cases of the project pilots, and implementation of semantic translation within their scope we believe that the above-presented set of transformations covers majority of practical needs. However, it should be stressed that the proposed approach is flexible and can easily be extended to capture additional practical needs.

5 Example

Let us now illustrate the usage of IPSM-AF format, by presenting a fragment of an alignment between a port ontology, that describes (among others) meteo stations and data they produce, and a suitably extended Generic IoT Platform Ontology (GOIoTP ontology[17]). The GOIoTP is a modular core ontology developed during the INTER-IoT project, which allows describing various aspects of IoT deployments, such as devices, platforms, observations, units and measurements, location, services, and users. GOIoTP imports and uses parts of several standard ontologies, such as SSN/SOSA,[18] GeoSPARQL,[19] and NASA SWEET units.[20] Since GOIoTP is a core ontology, it has been further extended to form the Generic Ontology for IoT Platforms Extended (GOIoTPex)—a vertical module that imports the GOIoTP ontology, and augments and "fills" selected stub concepts from GOIoTP with more specific classes, properties, and individuals. In short, GOIoTPex ontology extends GOIoTP with terms required in the concrete instantiations of the INTER-IoT solution. In particular, it serves as a central ontology for semantic translations executed within the INTER-IoT pilot applications, where both IPSM-AF and IPSM were utilized to provide a flexible and efficient semantic interoperability solution.

As discussed in [9], depending on the application/domain needs, practically any ontology can be selected as the central one, since the choice does not influence the semantic translation engine. However, semantic engineer should keep in mind that central ontology should: (i) cover all "topics" of conversations in the ecosystem, (ii) be clear enough to enable querying and reasoning done directly on it, and (iii) contain subject-specific modules that can be independently maintained and versioned. In this respect, GOIoTPex can be seen as an excellent "foundational" central ontology for any IoT-centric ecosystem.

[17] https://inter-iot.github.io/ontology/.

[18] https://www.w3.org/TR/vocab-ssn/.

[19] https://www.ogc.org/standards/geosparql.

[20] http://sweet.jpl.nasa.gov/2.3/reprSciUnits.owl.

Table 2 Prefixes used in the examples

Prefix	Namespace
	Source
port	http://inter-iot.eu/syntax/WSO2Port#
	Target
vp	http://inter-iot.eu/LogVPmod#
iiot	http://inter-iot.eu/GOIoTP#
iiotex	http://inter-iot.eu/GOIoTPex#
geo	http://www.opengis.net/ont/geosparql#
ogis	http://www.opengis.net/def/sf/
sosa	http://www.w3.org/ns/sosa/
meteo	http://www.inter-iot.eu/wso2port/weather/stations/

The example alignment consists of just two mappings (cells). Both refer to an ecosystem, in which meteorological data is important and, in addition, artifacts forming the ecosystem might be interested in data arriving from specific "registered" meteo stations. In the first mapping, we are interested in translating RDF messages exchanged within the ecosystem that represents meteo station "registration." The second mapping should enable translation of RDF graph structures into actual measurements coming from meteo stations. In both cases, we shall utilize the GOIoTPex as the target ontology. We assume that input data is represented in a (simple) *port* ontology expressing "flat" data coming from the port platform. Both examples (with some simplifications) originate from the INTER-IoT project pilot, in which data gathered by different IoT platforms is published and processed in an INTER-IoT ecosystem, deployed at the Port of Valencia, in Spain. The ontologies/prefixes used in the example alignment are listed in Table 2.

Let us start with the first example, in which a message describing a meteo station registration needs to be translated from the *port* ontology, and expressed in terms of the central ontology. In Listing 11, a meteo station called "P.Felipe" is characterized by a set of datatype properties (expressed using RDF Turtle notation).

```
[] a port:Element ;
port:haslatitude "26.94442"^^xsd:float ;
port:haslongitude "19.29351"^^xsd:float ;
port:hasmeteoStationId "2"^^xsd:int ;
port:hasname "P.Felipe" .
```

Listing 11 "P.Felipe" meteo station metadata

Alignment cell given in Listing 12 contains RDF graph patterns that generate correspondences (translation rules) that "match" any meteo station registration message, and produce its counterpart, expressed in terms of the GOIoTPex ontology.

```
<align:Cell rdf:about="&sripas;1_meteo_stations">
```

```
        <align:entity1 rdf:datatype="&xsd;string">
            var:elem a port:Element ;
                port:haslatitude var:lat ;
                port:haslongitude var:long ;
                port:hasmeteoStationId var:id ;
                port:hasname var:name .
        </align:entity1>
        <align:entity2 rdf:datatype="&xsd;string">
            var:station a vp:MeteoStation, iiot:IoTDevice, sosa:Sensor ;
                iiotex:hasLocalId var:id ;
                iiot:hasName var:name ;
                iiot:hasLocation [
                    a iiot:Location ;
                    geo:asWKT var:geopos
                ] .
        </align:entity2>
        <align:relation>=</align:relation>
        <sripas:transformation rdf:parseType="Literal">
            <function about="STR">
                <param order="1" about="&var;id"/>
                <return about="&var;sid"/>
            </function>
            <function about="CONCAT">
                <param order="1" val="&meteo;"/>
                <param order="2" about="&var;sid"/>
                <return about="&var;id_uri"/>
            </function>
            <function about="IRI">
                <param order="1" about="&var;id_uri"/>
                <return about="&var;station"/>
            </function>
            <function about="STR">
                <param order="1" about="&var;lat"/>
                <return about="&var;slat"/>
            </function>
            <function about="STR">
                <param order="1" about="&var;long"/>
                <return about="&var;slong"/>
            </function>
            <function about="CONCAT">
                <param order="1" val="Point("/>
                <param order="2" about="&var;slat"/>
                <param order="3" val=" "/>
                <param order="4" about="&var;slong"/>
                <param order="5" val=")"/>
                <return about="&var;geopos"/>
            </function>
        </sripas:transformation>
        <sripas:filters rdf:parseType="Literal">
            <filter about="&var;lat" datatype="&xsd;float"/>
            <filter about="&var;long" datatype="&xsd;float"/>
            <filter about="&var;slat" datatype="&xsd;string"/>
            <filter about="&var;slong" datatype="&xsd;string"/>
        </sripas:filters>
        <sripas:typings rdf:parseType="Literal">
            <typing about="&var;geopos" datatype="&ogis;wktLiteral"/>
        </sripas:typings>
    </align:Cell>
```

Listing 12 Alignment cell mapping meteo station “registration”

The message from Listing 11, when matched against the pattern from *entity1*, establishes appropriate “variable bindings” that are subsequently utilized/referenced in *entity2*, *transformation*, and *filters* sections of the cell. The structure of the RDF graph of the message together with the “variable bindings” is depicted in Fig. 1.

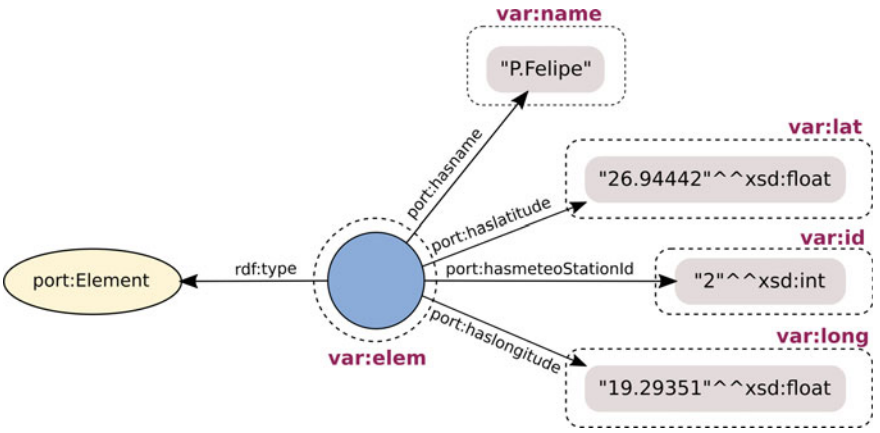

Fig. 1 RDF graph of the registration message

Because of the structure of the RDF graph pattern from *entity2* (Listing 12), and the created variable bindings, an instance of *vp:MeteoStation*, *iiot:IoTDevice*, and *sosa:Sensor* needs to be generated from the value that was matched by the *var:id* variable. This way, a numerical property of a blank node, representing a meteo station in the source data, is translated into an identifier (URI) of an entity representing the station in the target ontology. Therefore, in transformations sections, functions are called to cast it into string (STR), concatenate it with proper prefix (CONCAT), and generate the URI (IRI). The result is stored in the *var:station* variable that is referenced in the graph pattern from *entity2*. The source *port:hasname* property is mapped to the *iiot:hasName* property from the target (central) ontology.

The third important part of the mapping is defining the correspondences between geospatial data representations. Properties *port:haslatitude* and *port:haslongitude* are mapped onto *iiot:hasLocation* property, from GOIoTPex, with value being an instance of *iiot:Location* that has *geo:asWKT* property. Geospatial data, in the WKT format, needs to be computed (concatenated), which is done by applying the CON-

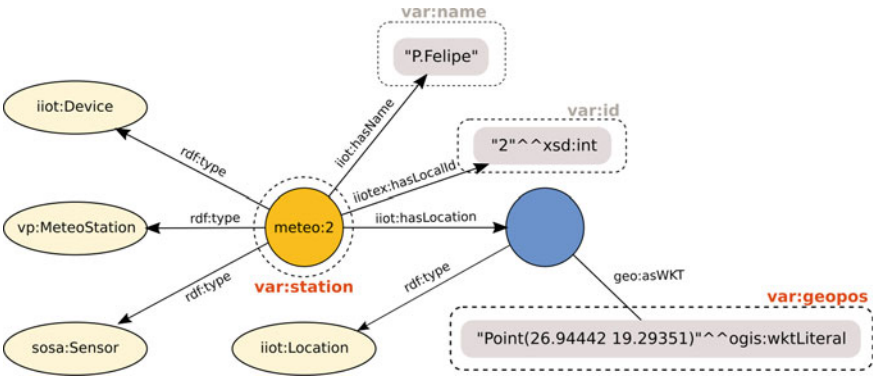

Fig. 2 RDF graph of the translated registration message and IPSM-AF variable bindings created by the alignment cell

CAT function that is called from the *transformation*. CONCAT takes as parameters string values; therefore, the STR function is called first, on source variables *var:lat* and *var:long*. Note that, in this case, *sripas:filters* and *sripas:typings* are used to include type filters and assign *ogis:wktLiteral* to the output variable. Figure 2 shows the structure of the translated RDF graph of the "registration" message, together with the variable bindings.

The result of the translation, represented in RDF Turtle notation, is given in Listing 13.

```
meteo:2 a vp:MeteoStation , iiot:IoTDevice , sosa:Sensor ;
iiot:hasLocation [
    a iiot:Location ;
    geo:asWKT "Point(26.94442 19.29351)"^^ogis:wktLiteral
] ;
iiot:hasName "P.Felipe" ;
iiotex:hasLocalId "2"^^xsd:int .
```

Listing 13 "P.Felipe" registration message after translation

The second example shows a mapping between meteorological data (observation) expressed in different semantics. Listing 14 shows sample RDF instance of an observation.

```
[] a port:Element ;
port:hasdate "2020-07-15T09:50:01.000Z" ;
port:hasmeasurementId "3181710"^^xsd:int ;
port:hasmeteoStationId "9"^^xsd:int ;
port:haspressure "1025.552"^^xsd:float ;
port:hasseaTemperature "0.0"^^xsd:float ;
port:haswindDirection "177.4603"^^xsd:float .
```

Listing 14 RDF instance representing meteorological observation

The instance, with a given numerical measurement identifier, is described with date and identifier of the station it originated from and includes measured values for pressure, temperature, and wind direction. The cell, for mapping observation data, is presented in Listing 15. The mapping defines how to transform input RDF graph, based on the *port* ontology, into an output RDF graph, based on the GOIoTPex ontology, with additional module for the meteorological data. The RDF graph structures, for the instance and the variable bindings that result from matching the alignment cell, are depicted in Fig. 3.

```
<align:Cell rdf:about="&sripas;1_weather_measurement">
    <align:entity1 rdf:datatype="&xsd;string">
        var:elem a port:Element ;
            port:hasmeasurementId var:meas_id ;
            port:hasmeteoStationId var:station_id ;
            port:hasdate var:date ;
            port:haswindDirection var:direct ;
            port:hasseaTemperature var:seatemp ;
            port:haspressure var:pressure .
    </align:entity1>
    <align:entity2 rdf:datatype="&xsd;string">
        var:station a vp:MeteoStation, iiot:IoTDevice, sosa:Sensor ;
            iiotex:hasLocalId var:station_id .
        var:measurment a sosa:Observation, vp:WeatherMeasurement ;
            iiotex:hasLocalId var:meas_id ;
```

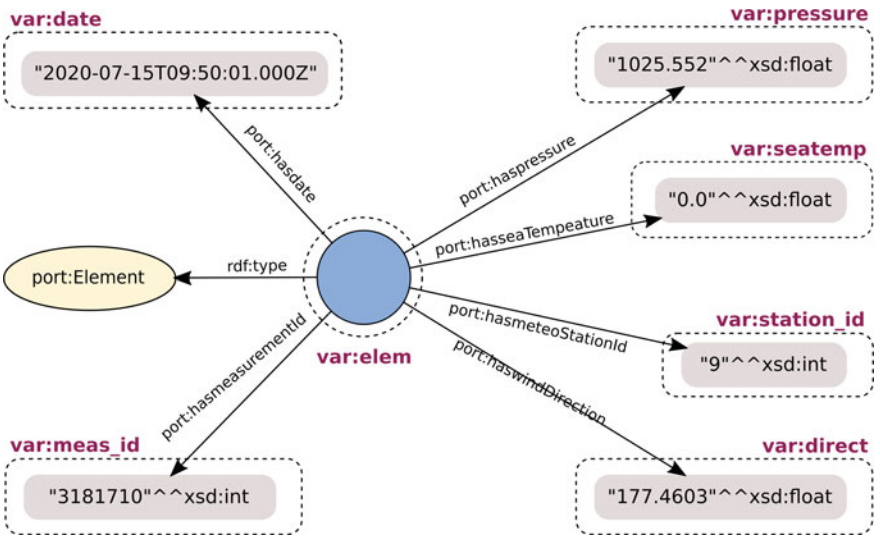

Fig. 3 RDF graph of the meteorological observation

```
            sosa:madeBySensor var:station ;
            sosa:hasResult [
                a sosa:Result, vp:WindDirection ;
                iiot:hasResultValue var:direct
            ], [
                a sosa:Result, vp:SeaTemperature ;
                iiot:hasResultValue var:seatemp
            ], [
                a sosa:Result, vp:Pressure ;
                iiot:hasResultValue var:pressure
            ] ;
            sosa:resultTime var:date .
    </align:entity2>
    <align:relation>=</align:relation>
    <sripas:transformation rdf:parseType="Literal">
        <function about="STR">
            <param order="1" about="&var;station_id"/>
            <return about="&var;sid"/>
        </function>
        <function about="CONCAT">
            <param order="1" val="&meteo;"/>
            <param order="2" about="&var;sid"/>
            <return about="&var;sid_uri"/>
        </function>
        <function about="IRI">
            <param order="1" about="&var;sid_uri"/>
            <return about="&var;station"/>
        </function>
        <function about="STR">
            <param order="1" about="&var;meas_id"/>
            <return about="&var;mid"/>
        </function>
        <function about="CONCAT">
            <param order="1" about="&var;id_uri"/>
            <param order="1" val="/"/>
            <param order="2" about="&var;mid"/>
```

```
            <return about="&var;meas_id_uri"/>
        </function>
        <function about="IRI">
            <param order="1" about="&var;meas_id_uri"/>
            <return about="&var;measurment"/>
        </function>
    </sripas:transformation>
    <sripas:typings rdf:parseType="Literal">
        <sripas:typing about="&var;date" datatype="&xsd;dateTimeStamp"/>
    </sripas:typings>
</align:Cell>
```

Listing 15 Alignment cell for translation of meteorological observations

Next, the alignment is applied to transform the data, and the resulting output is as in Listing 16.

```
<http://www.inter-iot.eu/wso2port/weather/stations/9/3181710>
    a sosa:Observation , vp:WeatherMeasurement ;
    iiotex:hasLocalId "3181710"^^xsd:int ;
    sosa:hasResult [
        a sosa:Result , vp:WindDirection ;
        iiot:hasResultValue "177.4603"^^xsd:float
    ] ;
    sosa:hasResult [
        a sosa:Result , vp:SeaTemperature ;
        iiot:hasResultValue "0.0"^^xsd:float
    ] ;
    sosa:hasResult [
        a sosa:Result , vp:Pressure ;
        iiot:hasResultValue "1025.552"^^xsd:float
    ] ;
    sosa:madeBySensor meteo:9 ;
    sosa:resultTime "2020-07-15T09:50:01.000Z"^^xsd:dateTimeStamp .

meteo:9
    a vp:MeteoStation , iiot:IoTDevice , sosa:Sensor ;
    iiotex:hasLocalId "9"^^xsd:int .
```

Listing 16 Output for translation of meteorological observation

In the translation, identifiers of measurement and meteo station are transformed from numerical values into entities using IRI function applied to the source variables identifiers concatenated with prefixes. Meteo station is an instance of *iiot:IoTDevice*, *sosa:Sensor*, *vp:MeteoStation*, whereas measurement is an instance of *sosa:Observation*, *vp:WeatherMeasurement*. An observation is annotated with a reference to the specific sensor (meteo station), by which it was taken, as well as a result time and three results with values and classes, which indicate what was actually measured (Fig. 4).

6 Concluding Remarks

With the growing popularity of IoT-based solutions, big data processing, and systems integration, the need arose not only to be able to express correspondences (alignments) between data models, but also to use them in practices to realize needed (pos-

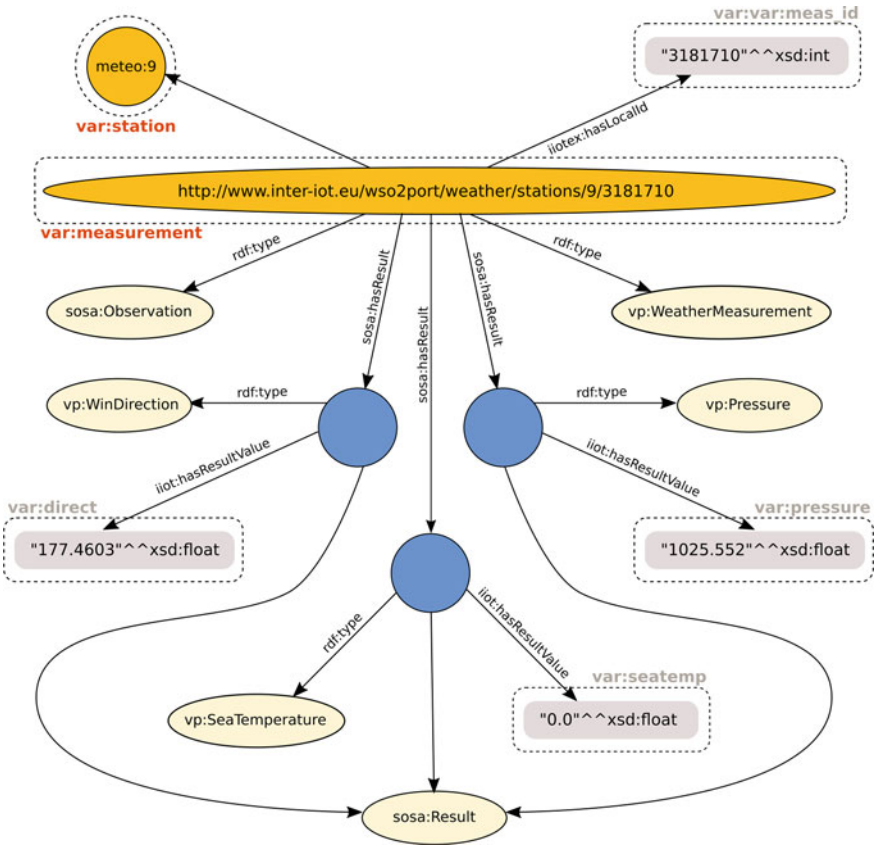

Fig. 4 RDF graph of the translated meteorological observation

sibly streaming) translations. Alignments should be persisted in a machine-readable format that can be automatically consumed by software tools, but is also easy to pick-up by humans. Here, IPSM-AF proves to be an expressive language for defining alignments that can be used by IPSM which automatically executes semantic translation. IPSM-AF was tested in two pilot applications and open call projects where different requirements for semantic translation needed to be fulfilled. Compliance with well-known Alignment API format enables alignments expressed in IPSM-AF to be consumed (if required after adjusting) by other semantic translators that may be developed in the future.

Outside of the validation of IPSM-AF in the INTER-IoT project [17], the format is supported by a rich number of concrete "recipes"[21] and guides for practical usage,[22]

[21] https://inter-iot.readthedocs.io/projects/inter-iot-cookbook/en/latest/inter-layer/ds2ds/recipes/alignment/.

[22] https://inter-iot.readthedocs.io/projects/inter-iot-cookbook/en/latest/inter-layer/ds2ds/appendices/products/.

including deeply analyzed examples, e.g., for mapping of geolocation data [6], as well as comparison with other possible alternatives [22].

The complete source code for the Inter-Platform Semantic Mediator (IPSM) can be freely obtained from the INTER-IoT official code repository[23] on GitHub. The easiest way to test/use IPSM is to follow the IPSM Docker image deployment instructions available from the INTER-IoT IPSM documentation site.[24]

Acknowledgments This research was partially supported by the European Union's "Horizon 2020" research and innovation program as part of the "Interoperability of Heterogeneous IoT Platforms" (INTER-IoT) project under the Grant Agreement No. 687283.

References

1. Borst WN (1997) Construction of engineering ontologies for knowledge sharing and reuse. Ph.D. thesis. University of Twente, Enschede, Netherlands. base-search.net (ftuni-vtwente:oai:doc.utwente.nl:17864)
2. Cousin P (ed) Internet of Things success stories, series 1-3. Internet of Things European Research Cluster (IERC) and Smart Action, 2014–2015
3. David J, Euzenat J, Scharffe F, Trojahn dos Santos C (2011) The alignment API 4.0. Semantic Web 2(1):3–10
4. Euzenat J, Shvaiko P (2013) Ontology matching, 2nd edn. Springer
5. Ferdinand M, Zirpins C, Trastour D (2004) Lifting XML schema to OWL. In: Koch N, Fraternali P, Wirsing W (eds) Web engineering—4th International Conference. ICWE 2004, Munich, Germany, 26–30 July 2004, Proceedings. Springer, Heidelberg, pp 354–358
6. Ganzha M, Paprzycki M, Pawłowski W, Szmeja P, Wasielewska K (2017) Alignment-based semantic translation of geospatial data. In: Proceedings of 3rd International conference on advances in computing, communication & automation (ICACCA)
7. Ganzha M, Paprzycki M, Pawłowski W, Szmeja P, Wasielewska K (2017) Semantic interoperability in the Internet of Things: an overview from the INTER-IoT perspective. J Netw Comput Appl 81:111–124
8. Ganzha M, Paprzycki M, Pawłowski W, Szmeja P, Wasielewska K (2017) Streaming semantic translations. In Proceedings of 21st international conference on system theory, control and computing ICSTCC. IEEE, pp 1–8
9. Ganzha M, Paprzycki M, Pawłowski W, Szmeja P, Wasielewska K (2017) Towards semantic interoperability between Internet of Things platforms. In: Gravina R, Palau CE, Manso M, Liotta A, Fortino G (eds) Integration. Interconnection, and interoperability of IoT systems. Springer, Cham, pp 103–127
10. Ganzha M, Paprzycki M, Pawłowski W, Szmeja P, Wasielewska K, Fortino G (2016) Tools for ontology matching—Practical considerations from INTER-IoT perspective. In: Proceedings of the 8th international conference on internet and distributed computing systems, vol 9864 of LNCS. Springer, pp 296–307
11. Ganzha M, Paprzycki M, Pawłowski W, Szmeja P, Wasielewska K, Palau CE (2017) From implicit semantics towards ontologies—Practical considerations from the INTER-IoT perspective (submitted for publication). In: Proceedings of 1st edition of globe-IoT 2017: towards global interoperability among IoT systems

[23] https://github.com/INTER-IoT/ipsm-core.

[24] https://inter-iot-ipsm.readthedocs.io/en/latest/Deployment/Docker-image/.

12. Ganzha M, Paprzycki M, Pawłowski W, Szmeja P, Wasielewska K, Solarz-Niesłuchowski B, de Puga García JS (2018) Towards high throughput semantic translation. In: Fortino G, Palau CE, Guerrieri A, Cuppens NCF, Chaouchi H, Gabillon A (eds) Interoperability, safety and security in IoT. Springer, Cham, pp 67–74
13. Heyvaert P, Dimou A, De Meester B, Seymoens T, Herregodts A-L, Verborgh R, Schuurman D, Mannens E (2018) Specification and implementation of mapping rule visualization and editing: MapVOWL and the RMLEditor. J Web Semantics 49:31–50
14. Kharlamov E, Kotidis Y, Mailis T, Neuenstadt C, Nikolaou C, Özçep Ö, Svingos C, Zheleznyakov D, Ioannidis Y, Lamparter S, Möller R, Waaler A (2019) An ontology-mediated analytics-aware approach to support monitoring and diagnostics of static and streaming data. J Web Semantics 56:30–55
15. Miles A, Matthews B, Wilson M, Brickley D (2005) SKOS core: simple knowledge organisation for the web. In: Proceedings of the 2005 international conference on dublin core and metadata applications: vocabularies in practice, DCMI'05. Dublin Core Metadata Initiative, pp 1–9
16. Mohammadi M, Rezaei J (2020) Evaluating and comparing ontology alignment systems: an MCDM approach. J Web Semantics 64:100592
17. Palau CE, Fortino G, Montesinos M, Exarchakos G, Giménez P, Markarian G, Castay M, Fuart F, Pawłowski W, Mortara M, Bassi A, Gevers F, Ibañez G, Huet I (eds) (2021) Interoperability of Heterogeneous IoT platforms—A layered approach. Internet of Things. Springer
18. Perera C, Liu CH, Jayawardena S, Chen M (2015) Context-aware computing in the Internet of Things: a survey on Internet of Things from industrial market perspective. In: CoRR. abs/1502.00164
19. Scharffe F (2009) Correspondence patterns representation. Ph.D. thesis. University of Innsbruck
20. Scharffe F, Zamazal O, Fensel D (2014) Ontology alignment design patterns. Knowl Inf Syst 40:1–28, 7
21. Szmeja P, Ganzha M, Paprzycki M, Pawłowski W, Wasielewska K (2018) Declarative ontology alignment format for semantic translation. In: 3rd International conference on Internet of Things: smart innovation and usages (IoT-SIU 2018). IEEE Xplore, pp 1–6
22. Szmeja P, Prud'hommeaux E (2021) ShExMap and IPSM-AF—comparison of RDF transformation technologies. In: Intelligent systems, technologies and applications, Proceedings of sixth ISTA 2020, India. Springer, Berlin, Germany, pp 29–46
23. Thiéblin E (2019) Automatic generation of complex ontology alignments. Ph.D. thesis. Institut de Recherche en Informatique de Toulouse, Toulouse, 10 2019
24. Thieblin E, Haemmerlé O, Hernandez N, dos Santos CT (2019) Survey on complex ontology matching. Semantic Web
25. Volz J, Bizer C, Gaedke M, Kobilarov G. Silk—A link discovery framework for the web of data. LDOW, 538