Declarative Ontology Alignment Format for Semantic Translation

Paweł Szmeja*, Maria Ganzha*‡, Marcin Paprzycki*§, Wiesław Pawłowski†*, Katarzyna Wasielewska*
* Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland
† Faculty of Mathematics, Physics, and Informatics, University of Gdańsk, Gdańsk, Poland
‡ Warsaw University of Technology, Warsaw, Poland
§ Warsaw Management Academy, Warsaw, Poland

***Abstract*—With the rise of Linked Data, triplestores, natural language processing and other semantic technologies, popularity and applicability of ontologies grows as well. One of more common operations on ontologies, regardless of their application, is an alignment – a matching between entities from different ontologies. So far, many alignment formats and languages have been proposed, some of them for general use and some particular to a concrete implementation of an alignment software. In this paper we present an ontology alignment format developed within the INTER-IoT project, and compatible with the Alignment API. Named after our semantic translation software – Inter Platform Semantic Mediator (IPSM) – the *IPSM Alignment Format* is universal and can be used to express both simple andf complex alignments. It is applied in practice, and used in semantic translations within INTER-IoT.**

I. Introduction

Development of semantic technologies can be viewed from the perspective of the "Garner Hype Cycle"[1]. Clearly, the time of "Peak of Inflated Expectations" followed publication of an influential book by D. Fensel [1]. At that time a number of volunteer-bases efforts attempted at finding ways of applying ontologies in practice; e.g. project ChefMoz which tried to build a "semantic restaurant repository"[2]. This was also the time EC funded a number of research projects in this area; see, for instance information concerning the Knowledge Web project[3]. However, when one looks at it from birds-eye view, expected progress has not been achieved. This resulted in "Trough of Disillusionment", when researchers have moved to other, funded, research areas. It is only recently when the interest in the topic re-emerged, giving hope that the "Slope of Enlightenment" is near. Here, the sings are, among others: rise of Linked Data, development of software to use triplestores, or advances on natural language processing – with use of semantic technologies. Nevertheless, these are just early beginnings of change. This can be seen, for instance, when one considers the state-of-the-art in tools for ontology alignments. In this context, in [2], we have shown that only about 10% of developed tools (described in literature and tried in the OEAI competition) are still "alive". Furthermore, in [3] we have illustrated – on a very simple, and rather straightforward, example – that their capabilities are somewhat limited.

Nevertheless, believing that the raise of semantic technologies is near and that they have an enormous practical potential, in the INTER-IoT project [4] we have decided to apply them to facilitate interoperability between IoT artifacts. Here the assumption is that each IoT artifact offers an ontology, and that it is possible to translate messages generated by such artifact into RDF triples representing their content in terms of the ontology (syntactic translation). Similarly, it should be relatively easy to translate incoming messages represented in terms of the ontology into the native format of a given artifact. In other words, it should be clear that translating messages from the native format to the ontology formally representing it (and back) can be achieved without serious complications. Obviously, it has to be assumed that ontologies of individual artifacts can (and are likely to) differ. Therefore, ontology aligning is needed to establish communication *between* artifacts.

There are multiple ways to find alignments between ontologies. It can be done using one of existing tools. However, as mentioned, they have not reached maturity. It is also possible to use the "paper and pencil" method and match ontologies "manually". In this context, we would like to ask: how to represent alignment(s) between ontologies. So far, many alignment formats and languages have been proposed, some of them for general use and some particular to a concrete implementation of an alignment software (see, Section II). Nevertheless, in this paper, we present our own ontology alignment format developed within the scope of the INTER-IoT project. This format is compatible with the Alignment API [5] and inspired by the EDOAL [6]. Named after our semantic translation software – Inter Platform Semantic Mediator (IPSM) – the IPSM Alignment Format is universal and can be used to store both complex and simple alignments. It is applied in practice and used in the process of semantic translation within INTER-IoT.

In what follows, we start (in Section II) with discussion of the state-of-the-art. This discussion provides arguments. why we have decided to develop our own approach to representing alignments. Next, in Section III, with the description of our approach. Further, technical details concerning the IPSM Alignment Format are summarized in Section IV.

[1] http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp
[2] https://en.wikipedia.org/wiki/ChefMoz
[3] http://cordis.europa.eu/ist/kct/knowledgeweb_synopsis.htm

II. Alignment Formats

The need for use of alignments arose during work on mechanisms of semantic translation of messages in INTER-IoT project. Therefore, we required a format that would be practical, extendable and capable of storing complicated expressions and transformations. There are few existing alignment formats, proposed in recent years. First and, probably, most popular is the Alignment API Format [5]. It is a standard RDF/XML format for storing alignments. It was popularized by the OAEI[4] competition, where it serves as the common output format, so that the results of different software tools can be directly compared. The schema of the format allows different "levels" of complexity. The simplest – level 0 (the default for the Alignment API) allows to express correspondences between simple entities, i.e. ones described only by an URI. At the same time there are no (known to us) tools that can support higher levels of the Alignment API. Therefore, since we had to represent alignments more complex than these that the level 0 Alignment API can capture, we have started looking for other options.

EDOAL [7], by the same authors as the Alignment API, is a declarative language (compatible with the schema of the Alignment API format) capable of expressing complex alignments. It defines its own OWL-like language to describe entities and transformations between them. Although, at the time of writing, there is no tool that can generate complex alignments (i.e. as complex as EDOAL allows) automatically, there were attempts at designing semi-automatic systems [8], [9]. There, the automatically recognized equivalence or subsumption matches between simple entities were expected to be manually refined, in an iterative process, in order to capture/represent existing complexity. While expressive, EDOAL has its own limitations, some even recognized by the authors [7]. In particular, it does not support RDF patterns in entities, instead opting for a custom RDF-like set of XML tags. Furthermore, its practical application requires learning a new way to write class and property expressions, relations, values, and others. Moreover, the EDOAL transformations are complicated and, in our opinion, not explained well enough in the documentation.

Some alignment tools, like COMA++[5] or AgreementMaker[6], alongside the capability of generating Alignment API format-compatible files, define their own alignment formats. Those are, however, completely application-specific and are not used outside of their respective software.

RDF itself, can be annotated using vocabularies, such as SKOS[7], in order to store the alignment information in the same file, as the ontology. However, this approach has been criticized for mixing ontology with extraneous data, and being too constrained [10] by the semantics of the underlying ontology language (i.e. OWL). Hence it also did not provide a viable alternative.

[4] http://oaei.ontologymatching.org/
[5] https://dbs.uni-leipzig.de/de/Research/coma.html
[6] https://github.com/AgreementMakerLight/AML-Jar
[7] https://www.w3.org/TR/2009/REC-skos-reference-20090818/

There are also other conceptual models of representing alignments, such as [10]. Although they offer clear theoretical advantages and have useful mathematical properties, they lack implementations. Hence, they had no practical value for the needs of the INTER-IoT project.

Taking into consideration our findings, we have decided to create our own format, as opposed to implement an existing specifications (with constraints put upon the format by its original authors). Let us now describe it in some detail.

III. IPSM Alignment Format

The general structure of the Inter Platform Semantic Mediator Alignment Format (IPSM-AF) is based on the Alignment API format [5], and is fully compatible with its specification. IPSM-AF was also inspired by the EDOAL [7], but has key differences and additions, both in syntax and semantics. The format is serialized in RDF/XML, although it can be stored in any RDF serialization. Here, note that the XML is required for compatibility with the Alignment API.

Conceptually, an IPSM-AF file describes a set of unidirectional mappings (alignments) between *cells*. Each cell is an independent structure describing an RDF graph. The alignment represents not just a simple correspondence, but also a way of transforming the "source" RDF into the "target" RDF code. Unidirectionality of mappings means that the correspondence holds between the source and the target cells, but not necessarily in the opposite direction. This is significant, because, in this way, it is possible to capture complicated alignments, which use transformations that may not be reversible.

```
<?xml version='1.0' encoding='utf-8' standalone='
    no'?>
<rdf:RDF xmlns="http://www.inter-iot.eu/sripas#"
  {other xml namespaces} >
<align:Alignment>
  <dcelem:title>{title}</dcelem:title>
  <exmo:version>{version}</exmo:version>
  <dcelem:creator>{creator}</dcelem:creator>
  <dcelem:description>{description}</
      dcelem:description>

  <align:xml>yes</align:xml>
  <align:level>2IPSM</align:level>
  <align:type>**</align:type>
  <align:method>{method}</align:method>
  <align:time>{time}</align:time>

  <align:onto1>
    <align:Ontology rdf:about="[source ontology
        uri]">
      <align:location>
        {source ontology location}
      </align:location>
      <align:formalism>
        <align:Formalism
          align:name="[src ontology formalism name
              ]"
          align:uri="[src ontology formalism URI]"
        />
      </align:formalism>
    </align:Ontology>
  </align:onto1>
  <align:onto2>
    {target ontology information}
```

```
  </align:onto2>

  <sripas:steps rdf:parseType="Literal">
    <sripas:step
      sripas:order="[cell order]"
      sripas:cell="[cell id]"
    />
    {more steps}
  </sripas:steps>

  <align:map>
    {alignment cell}
  </align:map>
  {more mappings}
</align:Alignment>
</rdf:RDF>
```

Listing 1. IPSM-AF — general structure

The overview of the structure of the IPSM-AF is presented in Listing 1. An alignment file is, conceptually, divided into a set of *header* properties and a set of *mappings* that contain information most important for the alignment. The header is a simple collection of properties and follows, mostly, the Alignment API. It contains metadata about the alignment. Here one can find property assertions, such as **<dcelem:title>**, **<exmo:version>**, **<dcelem:creator>** and **<dcelem:description>** that describe the contents of an alignment file as simple text (i.e. in strings). Values of those properties are also used in the INTER-IoT semantic repository (storage) software and the Alignment Server[8]. Because of inherent extendability of RDF, other properties, such as **<rdfs:comment>** or **<dcterms:issued>**, may be added.

Properties, inherited from the Alignment API, are as follows. **<align:xml>** states that the alignment file is formalized as RDF/XML and, for the IPSM Alignment Format, it should always be set to "yes". **<align:level>** is the Alignment API indicator of complexity level of the alignment, set to "2IPSM", for our format This name follows the example set in the Alignment API documentation, where "0" means simple entity alignments, "1" simple entity groups, and "2" complex entities (e.g. "2EDOAL" is the identifier for EDOAL). **<align:type>** describes the arity (e.g. 1-to-1, 1-to-many), which is "**" for our format (denoting many-to-many). The two properties: **<align:method>** and **<align:time>**, describe the method used to create the alignment, and the time when it happen, respectively. Reader should refer to the Alignment API documentation[9] for more in-depth explanation of these properties.

The header contains also information about *source* and *target* ontologies, as properties **<align:onto1>** and **<align:onto2>**, respectively – both inherited from the Alignment API. Technically, this declaration does not constrain the content of the alignment cells in any way. In theory, one could put, in the alignment file, mappings that do not use either ontology. Conceptually, however, the aligned ontologies should correspond with the "source" and "target" cells. It is up to the alignment creator to decide if the alignment will uphold this rule.

The last header property – **<sripas:steps>** – describes an "order of application" of alignment cells, which should be interpreted in the context of alignment-based semantic translation, and is used by the IPSM. We will get back to this in further sections. What is significant about this property is the use of the **rdf:parseType** attribute with the value **"Literal"**, which denotes that the object of the property is an XML literal. It increases the readability of the IPSM-AF file, and is a simple way of avoiding the "multiple children problem" that would arise, if a set of RDF property assertions and objects was used instead.

The **rdf:parseType="Literal"** declaration is also used in other properties, described later. Values of such properties still need to be in well-formed XML, following a separate schema, defined for XML and not RDF. For **<sripas:steps>**, the schema constraints the value of the property to a list of **<sripas:step>** tags with integer attribute **order** and string attribute **cell**.

Finally, objects of the **<align:map>** property are the actual alignment cells. Alignment can have any number of **<align:map>** property assertions, each with an **<align:Cell>** object as the value. Note that a useful alignment would have at least one cell. The structure of an alignment cell is presented in Listing 2.

```
<align:Cell rdf:about="[cell id]">
  <align:entity1 rdf:parseType="Literal">
    {source RDF pattern}
  </align:entity1>
  <align:entity2 rdf:parseType="Literal">
    {target RDF pattern}
  </align:entity2>
  <align:relation>{relation}</align:relation>
  <align:measure rdf:datatype="&xsd;float">
    {confidence measure}
  </align:measure>
  <sripas:filters rdf:parseType="Literal">
    <sripas:filter sripas:about="&sripas;node_sx"
        sripas:datatype="&xsd;string"/>
    {more filters}
  </sripas:filters>
  <sripas:typings rdf:parseType="Literal">
    <sripas:typing sripas:about="&sripas;node_y"
        sripas:datatype="&xsd;float"/>
    {more typings}
  </sripas:typings>
  <sripas:transformation rdf:parseType="Literal">
    {functional constraints}
  </sripas:transformation>
</align:Cell>
```

Listing 2. IPSM-AF – alignment cell structure

In the listing, the two most important properties are **<entity1>** and **<entity2>** that describe *source* and *target* entities, respectively. They contain the RDF/XML code, with an *RDF pattern*, which is any valid RDF code, optionally with additional sripas tags (explained below). Here, the **rdf:parseType="Literal"** declaration is used, so that the property values are treated as literals. As a consequence the RDF graphs, inside the literals, are separated from the RDF graph of the whole alignment. In other words, the source

[8] http://alignapi.gforge.inria.fr/server.html
[9] http://alignapi.gforge.inria.fr/format.html

and the target entity RDF patterns are kept encapsulated and separate.

```
<align:entity1 rdf:parseType="Literal">
  <sripas:node_X>
    <rdf:type rdf:resource="&dc;Agent"/>
  </sripas:node_X>
</align:entity1>
```

Listing 3. IPSM-AF – example of source RDF pattern

Listing 3 presents a simple RDF pattern, with a single **<sripas:node_X>** *variable*. The `sripas` is a reserved namespace that has a special meaning in the RDF pattern. It denotes an element that is interpreted as a "placeholder", or a variable, and its name is the URI fragment (e.g. `node_X` in the example in Listing 3). Variable names do not need to be defined before their use and there can be any number of them, used anywhere in the pattern. Multiple patterns may use the same variables. Within an alignment cell, there are no limitations put on variables in **<entity1>**. Variables in **<entity2>**, however, must be declared earlier in **<entity1>**, or in **<transformation>** (described later). In other words, one cannot use a variable in **<entity2>** that did not appear earlier in the same alignment cell.

Observe that meaning of `sripas` variables is very similar to SPARQL variables – they are placeholders for any object or value. In Listing 3, for example, the `node_X` variable denotes any valid RDF node that is the subject of a "`?node_X rdf:type dc:Agent`" triple. In this example, the variable represents an object, but the same mechanism can be used to represent literal values (just like in SPARQL). At times, this may be in opposition to semantics of a particular ontology, because a `sripas` variable is, technically, a node for a named entity in the `sripas` namespace. Putting it in the place of the object of a triple, where the predicate is an OWL *data* property, violates the semantics of some OWL profiles (e.g. OWL 2.0 DL). The semantics of our format, however, allows such RDF pattern. Otherwise the *value* type and *object* type variables would need to have different syntax, which we wanted to avoid.

In order to accommodate added semantics, while preserving RDF compatibility, the datatype of variables cannot be restricted from within RDF patterns. Instead, each cell can have optional **<sripas:filters>** and **<sripas:typings>** properties. The **<sripas:filters>** restrict datatypes of variables from the "source", i.e. **<entity1>**. The value of the property is a literal, with a collection of **<filter>** tags, each with **sripas:about** and **sripas:datatype** attributes. Those tags are interpreted as restricting variables, so that they must be of a given RDF datatype, in order to match the *source* entity. The **<sripas:typings>** property states (i.e. declares) the RDF datatypes of variables in the *target* entity. The value of the property is a set of **<sripas:typing>** tags with the same attributes, as the **<sripas:filter>**. At this point *typings* and *filters* pertain only to variable datatypes.

```
<align:entity2 rdf:parseType="Literal">
  <sripas:node_X>
    <rdf:type rdf:resource="&sc;Person"/>
  </sripas:node_X>
</align:entity2>
```

Listing 4. IPSM-AF – example of target RDF pattern

Listing 4 contains a simple example of a *target* entity RDF pattern that could be paired with the *source* from Listing 3. Here, **<entity2>** makes use of the `node_X` variable declared in *source*. Provided the cell does not use any other information (i.e. typings, filters or transformations) the meaning of this alignment is as follows – the type `dc:Agent` of any entity is aligned with type `sc:Person`. The variable is the reference point that lets us state: any entity of type `dc:Agent` in *source* semantics is of type `sc:Person` in *target* semantics.

The **<align:relation>>** property denotes the type of correspondence between **<entity1>** and **<entity2>**, e.g. subsumption (<), supersumption (>) and equivalence (=). Even though "=" is symmetric, IPSM alignments should always be treated as uni-directional, because of additional transformations that may have no reverse transformation defined (more, in what follows). If a cell does not have any transformations, typings and filters, then it can be trivially reversed by swapping contents of *source* and *target* entities.

The **<align:measure>** property denotes confidence level between 0 and 1. It is, usually, produced by software that made the alignment. The larger the value, the more sure we are that the correspondence between entities holds true.

Finally the **<sripas:transformation>** property (in Listing 5) contains any number of transformations that need to be performed in order to convert **<entity1>** RDF pattern into **<entity2>** pattern.

```
<sripas:transformation>
  <sripas:function sripas:about="[function name]">
    <sripas:param sripas:order="[parameter number]
        " sripas:about="[sripas variable name]"/>
    <sripas:param sripas:order="[parameter number]
        " sripas:val="[parameter value]"/>
    {more params}
    <sripas:return sripas:about="[sripas variable
        name]"/>
  </sripas:function>
  {more functions}
</sripas:transformation>
```

Listing 5. IPSM-AF – functional constraints

Currently, the only allowed transformations are functions, described in **<function>** tags. Each function has a name (in **sripas:about** attribute), a number of (ordered) parameters (**<sripas:param>** tag) and a single return value (**<sripas:return>** tag). While any functions always returns the result into a `sripas` variable, the parameters can be either a "hardcoded" value, or a value stored in a `sripas` variable. Because functions can declare new variables (as "returns"), they can be concatenated together, i.e. return variables of any function can be used as input(s) in others. The **sripas:order** stores an 1-based index of a parameter (in case a function accepts more than one), and **sripas:about** and **sripas:val** contain a reference to a variable, or a value respectively. Function names, their parameter requirements

and return values correspond directly to functions available in SPARQL. We revisit this in section IV.

Function, in Listing 6, is equivalent to the SPARQL `BIND STR(?node_Y) as ?node_FY`. The alignment cell states: any entity with `sc:name` property assertion (with `node_Y` value) is mapped to an entity with `co:prop` property assertion, whose value is the result of `STR` function run on `node_Y`.

```
<align:Cell rdf:about="&sripas;cell1">
  <align:entity1 rdf:parseType="Literal">
    <sripas:node_X>
      <sc:name>
        <sripas:node_Y/>
      </sc:name>
    </sripas:node_X>
  </align:entity1>
  <align:entity2 rdf:parseType="Literal">
    <sripas:node_X>
      <co:prop1>
        <sripas:node_FY/>
      </co:prop1>
    </sripas:node_X>
  </align:entity2>
  <sripas:transformation rdf:parseType="Literal">
    <sripas:function sripas:about="STR">
      <sripas:param sripas:order="1"
        sripas:about="&sripas;node_Y"/>
      <sripas:return sripas:about="&sripas;node_FY
          "/>
    </sripas:function>
  </sripas:transformation>
</align:Cell>
```

Listing 6. IPSM-AF – function example

IV. IPSM-AF TECHNICAL DETAILS

Bringing entity patterns as close to a regular RDF as possible, as well as using SPARQL functions in transformations, allowed us to leverage preexisting knowledge and experience with those technologies to create a format that is relatively familiar. Its structure exposes the primary use case, which is the semantic translation in INTER-IoT. As stated in the Introduction, we assume that at least two IoT artifacts (platforms, devices, services, etc.) are communicating with messages that can be syntactically transformed to and from RDF. The semantics of messages (i.e. ontologies used to describe entities) are different, therefore they need to be translated for every receiver.

Observe that, in the strictly technical sense, the semantic translation requires rewriting (i.e. replacing) subgraphs in the RDF messages, and IPSM-AF describes exactly how to do that. Each alignment cell is applied to the RDF graph and if it contains (i.e. matches) the **<entity1>** RDF pattern, along with any **<filters>**, then it is replaced with the RDF subgraph described by **<entity2>** RDF pattern, after application of **<transformation>**s and **<typings>**. In IPSM, cells are applied to the *source* message, in the order specified by the **<sripas:steps>** property. It allows declaration of an arbitrary ordering of application of alignment cells, where a given cell may even be used multiple times. This may be useful in rare cases where application of a specific cell depends on another cell being a match or not (an "if-type" situation in the alignment process).

This RDF rewriting itself is done by SPARQL UPDATE queries, to which the format is compiled by the IPSM software. The compiled queries use BIND statements to bind `sripas` variables, and apply any SPARQL functions declared in **<transformation>**. Other than the standard set of SPARQL 1.1 functions[10], custom user functions may be defined as extensions, or IPSM function libraries (powered by Apache Jena[11]). For more details about IPSM, see [11].

Working so close to the RDF brings forward important differences in interpretation of the content of alignments between the IPSM-AF format and others. Interpretation of RDF graph rewriting is very simple – we define, which graph is the "input", and which one is the "output". The rest of the alignment defines how to get from one to the other. This is different from, for example, the Alignment API or the EDOAL, where one can declare alignment between classes in a single alignment cell. The interpretation, in terms of RDF, is not officially defined, but we can imagine that, in case of 100% confident class equivalence, any RDF triple that uses *source* class URI could use *target* class URI instead, without change in semantics. In the IPSM-AF three cells are needed to achieve the same result – one for each: subject, predicate and object of RDF triples.

V. CONCLUDING REMARKS

In this work we have discussed in detail a new (the IPSM-AF) format for expressing alignments of RDF graphs, as well as any entities in any language expressible in RDF. Discussed approach is applicable, in particular, to any profile of OWL 2.0. The format is capable of storing very simple, as well as very complex alignments, and is compatible with existing technologies (i.e. Alignment API).

The proposed format is the result of work in the INTER-IoT project, where it is used in the process of semantic translation of messages exchanged between IoT artifacts. In this context, note that a concrete application of IPSM-AF and software to translation of geospacial data has been described in [3]. Furthermore, for more details about the IPSM we invite readers to see [12], [11], [12], [13], [14], [15], [16].

Given its practical usage in INTER-IoT we plan to develop CASE tools that would support users in creating alignments in our format, in order to remedy its complexity and enable its full potential. We will report our progress in subsequent publications.

VI. APPENDIX

In this paper we used the following RDF prefixes:

```
owl:    <http://www.w3.org/2002/07/owl#> .
rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
xsd:    <http://www.w3.org/2001/XMLSchema#> .
sripas: <http://www.inter-iot.eu/sripas#> .
```

[10] https://en.wikibooks.org/wiki/SPARQL/Expressions_and_Functions#Functions

[11] https://jena.apache.org/documentation/query/extension.html

```
exmo:   <http://exmo.inrialpes.fr/align/ext/1.0/#> .
dc:     <http://purl.org/dc/terms/> .
dcelem: <http://purl.org/dc/elements/1.1/> .
sc:     <http://schema.org/>
co:     <http://inter-iot.eu/central> .
align:  <http://knowledgeweb.semanticweb.org/
    heterogeneity/alignment#> .
```

Listing 7. Prefixes used

ACKNOWLEDGMENT

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

REFERENCES

[1] D. Fensel, *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Berlin: Springer-Verlag, 2000.

[2] M. Ganzha, M. Paprzycki, W. Pawłowski, P. Szmeja, K. Wasielewska, and G. Fortino, "Tools for ontology matching—practical considerations from INTER-IoT perspective," in *Proc. of the 8th Int. Conference on Internet and Distributed Computing Systems*, ser. LNCS, vol. 9864. Springer, 2016, pp. 296–307.

[3] M. Ganzha, M. Paprzycki, W. Pawłowski, P. Szmeja, and K. Wasielewska, "Alignment-based semantic translation of geospatial data," in *3rd International Conference on Advances in Computing, Communication & Automation (ICACCA), Proceedings*, in press.

[4] "INTER-IoT Project," http://www.inter-iot-project.eu.

[5] J. Euzenat, "An api for ontology alignment," in *International Semantic Web Conference*, vol. 3298. Springer, 2004, pp. 698–712.

[6] "EDOAL: Expressive and declarative ontology alignment language," http://alignapi.gforge.inria.fr/edoal.html.

[7] J. David, J. Euzenat, F. Scharffe, and C. Trojahn dos Santos, "The Alignment API 4.0," *Semantic Web*, vol. 2, no. 1, pp. 3–10, January 2011.

[8] D. Ritze, C. Meilicke, O. Šváb-Zamazal, and H. Stuckenschmidt, "A pattern-based ontology matching approach for detecting complex correspondences," in *Proceedings of the 4th International Conference on Ontology Matching-Volume 551*. CEUR-WS. org, 2009, pp. 25–36.

[9] F. Scharffe, J. Euzenat, Y. Ding, and D. Fensel, "Correspondence patterns for ontology mediation," in *Proceedings of the Ontology Matching Workshop at ISWC*, 2007.

[10] M. Pietranik and N. T. Nguyen, "A multi-attribute based framework for ontology aligning," *Neurocomputing*, vol. 146, no. C, pp. 276–290, 2014.

[11] M. Ganzha, M. Paprzycki, W. Pawłowski, P. Szmeja, and K. Wasielewska, "Streaming semantic translations," in *21st International Conference on System Theory, Control and Computing ICSTCC, Proceedings*, in press.

[12] ——, "Semantic interoperability in the Internet of Things: an overview from the INTER-IoT perspective," *Journal of Network and Computer Applications*, vol. 81, pp. 111–124, March 2017.

[13] ——, "Semantic technologies for the IoT – an Inter-IoT perspective," in *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*. Berlin, Germany: IEEE, April 2016, pp. 271–276.

[14] ——, "Towards semantic interoperability between Internet of Things platforms," in *Integration, Interconnection, and Interoperability of IoT Systems*, R. Gravina, C. E. Palau, M. Manso, A. Liotta, and G. Fortino, Eds. Springer, 2017, pp. 103–127.

[15] M. Ganzha, M. Paprzycki, W. Pawłowski, P. Szmeja, K. Wasielewska, and C. E. Palau, "From implicit semantics towards ontologies—practical considerations from the INTER-IoT perspective (submitted for publication)," in *Proc. of 1st edition of Globe-IoT 2017: Towards Global Interoperability among IoT Systems*, 2017.

[16] M. Ganzha, M. Paprzycki, W. Pawłowski, P. Szmeja, and K. Wasielewska, "Towards common vocabulary for IoT ecosystems—preliminary considerations," in *Intelligent Information and Database Systems, 9th Asian Conference, ACIIDS 2017, Kanazawa, Japan, April 3-5, 2017, Proceedings, Part I*, ser. LNCS, vol. 10191. Springer, 2017, pp. 35–45.