From implicit semantics towards ontologies—practical considerations from the INTER-IoT perspective

Maria Ganzha*‡, Marcin Paprzycki*, Wiesław Pawłowski†*, Paweł Szmeja*
Katarzyna Wasielewska*, Carlos E. Palau§
* 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
§ Departamento de Comunicaciones, Universitat Politècnica de València, Valencia, Spain

***Abstract*—From the general SOA architectural pattern, through distributed computing based on Grids and Clouds, to the Internet of Things, the idea of collaboration between software entities, independent from their vendors and technologies, attracts much attention. This brings about a question: how to achieve interoperability among multiple (existing and upcoming) platforms/systems/applications. The context for the presented research is provided by the INTER-IoT project, which deals with different aspects of interoperability in the Internet of Things (IoT). It aims at the design and implementation of an open framework and associated methodology to provide interoperability among heterogeneous IoT platforms, across a software stack (devices, network, middleware, application services, data and semantics). We focus on the data and semantics layer. Specifically, the role of ontologies and semantic data processing, as means of achieving interoperability. However, since the vision of the Semantic Web remains mostly unfulfilled, semantics remains implicitly "hidden" in data and in exchanged messages. Therefore, we are particularly interested in establishing: what methods and tools exist to create OWL ontologies from implicitly expressed semantics. We focus on popular data formats i.e. XML, JSON, RDF, Relational Databases and No-SQL Databases.**

I. Introduction

The *Internet of Things* (IoT) [24], conceptualized as an omnipresent network, consisting of physical and/or virtual objects, equipped with sensing and actuating capabilities, can be seen as an incarnation of, so called, *ubiquitous computing* [38], [26]. With billions of "things", combined into domain-specific ecosystems, the vision of the *hyper-connected world* is becoming a reality. Here, a problem of interoperability becomes a serious issue. While the question, how to make IoT platforms communicate, can be addressed on different levels, we are interested in data and semantics interoperability, which should enable "common understanding" of data and information.

The problems to overcome originate from different semantics used to describe domains of interest, and different data formats used to persist and exchange information. Here, we assume that there is only minimal incentive for "anyone" to make changes in their own platforms to make them interoperable. Moreover, we assume that more than two artifacts are to "work together". This makes approaches, based on building dedicated 1-to-1 solutions, unappealing (due to scalability problems). Therefore, we propose that platforms will interoperate through a "common semantics", i.e. a model that takes into account semantics of all participating systems (for more details of proposed approach, see [28]).

For the time being, the vision of the Semantic Web remains mostly a vision, and messages transmitted within and across IoT systems are not linked to a machine-processable ontological model. Therefore, typically, the semantics of existing applications is only implicitly represented. This being the case, the problem of finding relations between heterogenous semantics, represented as OWL ontologies (W3C Web Ontology Language [12]; discussed in [29]), becomes secondary. First, one has to "extract" semantics from data representations (data structures) used in applications (e.g. JSON, XML, RDF, relational and non-relational databases). We call this process "lifting" to OWL. Establishing what methods should be used to achieve this goal and what tools are available for most common data formats is the focus of this contribution. Finally, let us note that our approach is very pragmatic. We are interested only in existing and working tools and methods that have been tried .

II. INTER-IoT use case scenarios

The INTER-IoT project's main goal is to design and implement a framework that will allow interoperability between heterogeneous Internet of Things platforms, also across the *domain silos*. To ground our work, let us briefly discuss a use case scenarios based on a real world situation encountered by the project partners. Among the application domains considered by INTER-IoT pilots are: transportation and logistics (T&L) in a port domain, and mHealth. The aim of T&L use cases is to facilitate interoperability between IoT systems, operating in a large international port environment. Examples of IoT artifacts that are to work together are: road haulier platform, container terminal system, truck owner's platform, port management system, dynamic lighting platform, etc. The mHealth domain use cases are focused on interoperability

between IoT platforms for patient monitoring. One of them is based on a body sensor network, while the other works with wearable and non-wearable devices. In both cases, IoT artifacts are built using different technologies and use different data formats for storage and communication. Moreover, they use different semantics (encoded in data structures).

In order to establish a common semantics, and to relate semantics of each artifact to that of the others, we have to represent them in the same language (e.g. OWL), and, preferably, format (e.g. XML/RDF). This will allow us to use methods discussed in [29] to create an alignment between ontologies, or to merge them into a single ontology. Regardless of the specifics of this step, before it is attempted, we have to establish how to lift individual implicit semantics to their explicit semantic representation. Here, let us note that in the INTER-IoT project, we have decided to represent knowledge with OWL ontologies expressed in the RDF format. However, this decision has no effect on the generality of reported results.

III. Lifting to OWL ontologies

Let us now discuss the methods and tools available for lifting popular data formats to ontologies. We researched tools that may assist us in obtaining an OWL ontology. Here, an input is to be a data source or a schema, from which the information about semantics is extracted. The resulting ontology is "good" if the semantics of both input and output of the lifting process are close (ideally, they are identical). For other additional factors that refer to ontology quality see e.g. [21], [1], [2]. In what follows, we consider widely used data communication and storage formats including XML, JSON, RDB, non-relational databases. As will be shown, each format involves its own set of challenges and caveats.

A. XML

The XML is widely accepted human- and machine- readable structured textual data format used for storage, but mostly for exchange of information. Although it enables the systems to agree on a common syntax to effectively exchange information, the information may be misinterpreted because of different conceptualization of the modeled application domain. The original method to define the XML document structure is by using a Document Type Definition (DTD) that specifies the legal building blocks of the XML document as a list of elements and attributes along with their ordering and nesting. Another method to model the document structure is by using the XML Schema (XSD) that is written in XML, and provides more powerful means to define the constructs and their limitations (e.g. supports data-types and namespaces). It can be easily observed that both the DTD and the XSD can be mapped onto an OWL ontology with a conceptual model, while XML instances can be mapped onto individuals build on that ontology.

Some work has been devoted to the design and implementation of mechanisms for transformation of XSD into OWL (specifically into the RDF/RDFS syntax). In [34], JXML2OWL framework is discussed. It supports interactive and manual mappings and transformation from the syntactic data sources in the XML format, to a common shared global semantic model defined in the OWL. Specifically, the tool supports mappings between any XML schema (XSD or DTD) to classes/properties of an OWL ontology. Based on these mappings, the tool generates mapping rules as the XSLT stylesheets that allow automatic transformation of the XML data into instances of the mapped ontology. JXML2OWL offers both a Java library and a GUI application, and provides a valuable input to available methods. However, it requires the existence of explicit XSD or DTD, to transform XML instances. Moreover, mapping is done manually, which may be problematic for large ontologies with complex structures.

In [17], authors present a tool for mapping XML to OWL with an implementation based also on XSLT stylesheets — complete translation from an XML instances document, over a (possibly generated) XML Schema, to an OWL model with OWL instances. When the XML Schema is present, it is transformed to OWL. Otherwise, a schema generation from the XML instances is proposed (also based on the XSLT stylesheet). The transformation is based on rules for mapping XML element/attributes to ontology instances/properties and respective XML structures to OWL constructs. The tool XML2OWL is available online for testing, where it can be run for sample XML documents. However, at the moment of writing this paper (August, 2016), it is not working for any provided sample.

In [20], a direct mapping from the XML Schema to OWL, and from the XML to the RDF graph is proposed. However, these mappings are independent, so the resulting instances do not have to be consistent with the created ontology. Furthermore, potentially problematic situations, e.g. preserving order of XML elements, are not considered. Moreover, a tool realizing this approach does not seem to be maintained.

In [30], a more elaborate solution, dealing with more complex XML constructs (e.g. reuse of global types and elements), is described. The XML Schema is automatically translated to the ontology and mapping bridges are generated for translation of XML entities and for query translation. Authors stress that the second step (after running the algorithm), is a manual refinement of obtained ontology and of the mapping bridges. We have not found the described tool, however, we may treat related publications as a source of knowledge on how complex XML structures can be mapped onto OWL constructs.

In [16], a framework for generating OWL from a set of XML Schemas, based on predefined complex transformation patterns, is proposed. The tool is not available online, however the authors include numerous transformation patterns that can, for instance, be used for testing other solutions.

Authors of [40], describe an approach that deals with multiple XML data sources and design patterns, where the schema can be automatically generated. The mapping is based on the XML schema and the XML Schema Graph. We could not locate the tool described in the publication, so this work can be treated only as a source of theoretical knowledge.

Note that the problem with transforming "full" syntactic

specification from XML to OWL, has not been fully solved. For instance, XML constructs such as *sequence* and *choice* elements cannot be directly expressed in the OWL syntax. As a result, the information captured in ontologies does not always allow for the transformation of OWL descriptions to XML documents valid with respect to a given schema. This problem was addressed by authors of [36] where a framework for bidirectional transformations based on XS2OWL mapping model [37] is described. XS2OWL takes as input XSD and produces: (i) main ontology that captures XSD semantics, (ii) XML Schema with simple datatypes from the original ontology, (iii) mappings of OWL ontology construct ids to XSD construct names, and XSD constructs that cannot be represented in OWL. The XS2OWL XSLT stylesheet is available online, however we could not localize the full framework. Note that bidirectional translations are crucial to our work, which makes this work of value; regardless, the "missing tool".

Another very significant innitiative, described in [31], was undertaken by National Institute of Standards and Technology (NIST). A set of C++ software tools (part of them automatically generated) for manipulating XSD and XML instance files, and transforming them into OWL conceptual model files and OWL instance files was proposed. Note that in the proposed solution, XSD file needs to conform to some restrictions for the transformation process to be efficient and to overcome the problem of lack of full compatibility between XSD and OWL. Unfortunately, it seems that the tools are not publically available. Nevertheless, the paper contains throughout analysis of the steps required to perform the translation and problems that can arise.

Besides the aforementioned research, there are tools for XSD and XML data transformation available online including: Ontomalizer and ReDeFer. Ontomalizer allows automatic conversion based on a set of predefined rules, even for very large schemas, e.g. the HL7 CDA (one of the test cases). ReDeFer is a set of utilities for different transformations, e.g. XSD to OWL (based on the XSLT transform) and XML to RDF, that can be tested in an online translation service. There is also research devoted to transformation of DTD to OWL, including [14].

B. JSON

The *JavaScript Object Notation* (JSON), designed by Douglas Crockford, and defined in the IETF RFC 7159 [13], is a lightweight key-value type data-interchange format. Inspired by the JavaScript standard object literal syntax, it is currently one of the most widely used data formats, closely behind the decade-older XML.

Following the JavaScript *dynamic object* approach, JSON does not itself provide any kind of schema-capabilities. Unlike the XML, it does not currently have any standardized *schema language* either. The most popular attempt to provide one, so far, is the JSON Schema [27], [33]. Even though, it is still at the *IETF draft* level, its usage and tool support is growing. The JSON Schema, similarly to the XSD, uses the "object-language", i.e. JSON in this case, for defining schemas.

Regardless of its novelty, there are various tools supporting the JSON Schema format, such as validators, data-parsing libraries, and schema generators. Many of them have been listed on the language's main site [8]. Most interesting and useful, from the semantic point of view, are the *generators*. They allow to construct a schema, based on a single, or multiple, JSON document(s). Among the available generators let us mention: the highly configurable JSONSchema.net [9], a group of Node.js based tools, including [5], [3], and a Python based GenSon [6]. All of them analyze the structure of the input JSON document(s) to establish names of attributes and types of their values. The tools capable of accepting multiple input documents can also detect the optional and required attributes. Unfortunately, some of the more advanced features of the JSON Schema format, such as *validation keywords*, restricting the possible values of the attributes, are usually not fully taken into account by the generators. An interesting feature of the GenSon is that it can not only allows to input multiple JSON documents, but also multiple JSON schemas. All the input documents have to validate under the generated schema. Moreover, any JSON which is valid for all the input schemas must also validate against the generated one.

The JSON Schema documents can further be transformed into the XML Schema (XSD) format, although the choice of supporting tools is rather limited (e.g. [4], [10]). From the XSD schema, any of the methods mentioned above can be used to obtain an OWL ontology describing semantics of the data. At the same time, there are almost no methods of "directly" converting JSON Schema to OWL. One of the exceptions is an approach, based on the idea of *model-driven transformations*, presented in [39]. The authors have also created a prototype implementation of the method, but it does not seem to be available for the general use.

Note, that there is an intrinsic discrepancy between the JSON Schema and the XML Schema, which any conversion tool has to take into account. The latter describes a "closed content" where instances can contain only items explicitly allowed by the schema. The JSON Schema, on the other hand, specifies an "open content". Thus, schema instances—if not explicitly forbidden—may contain items beyond the ones requested by the schema. Therefore, while transforming the JSON Schema to the the XSD, one needs to make it open (e.g., by using the <any> element, wherever the JSON Schema does not prohibit additional properties).

The JSON format has also been used for serialization of *linked data*. The W3C JSON-LD recommendation can be seen as effectively defining a JSON serialization of RDF, although it allows some extensions, like *blank nodes* as predicates for example (RDF predicates have to be IRIs). Fortunately, these differences between the JSON-LD and the RDF can be treated in a uniform way (some suggested solutions can even be found in the recommendation), and standard tools like OWL API [11], [7] are capable of converting JSON-LD documents to RDF/XML, and hence also to OWL.

C. Relational databases

Relational databases (RDB) are, by far, the most popular way of storing data. Despite their legacy (or perhaps because of it), relational database models can be expressed in a variety of ways. Other than modeling artifacts, like Entity-Relationship diagrams, relational models can be formally written as a series of SQL statements. Unfortunately, SQL implementations are vendor-specific, and, even though common standards are largely adhered to, crucial details may not be compatible across different engines. Vendor-specific statements are usually not taken into account and the common SQL language base provides a relatively limited amount of structural information (compared to an ontology).

Data, in RDBs, is stored in tables with typed-columns. Entities in tables are identified by special subsets of columns called primary keys. Inter-table relations are stored as foreign keys, which contain information about direct one-to-one link between a subset of columns from two tables. Foreign keys do not need to be named (i.e. labeled). More precisely, explicit declaration of primary and foreign keys is usually optional. This means that it is very likely that a database will not contain any formal information about inter-relationships of entities stored within it. Also of note is the fact that relational databases do not declare any hierarchy of tables (an "approximation" of a taxonomy). In comparison, OWL individuals are identified by a globally unique URI and often have multiple types. Relations are always named, and taxonomies are often deep and wide. Ontologies form rich graphs of densely interlinked entities.

The characteristics of RDBs make the relational models distinctly different from OWL and, thus, difficult to convert in a way that is not only formally correct, but also yields useful and high quality ontologies. Nevertheless, a number of approaches exists that attempt to solve this problem. Even if the resulting ontology is not up to par, it can be a good starting point in conversion of relational model into an OWL ontology.

DB2OWL [19] is a tool for automatic construction of OWL DL ontologies from relational databases. It inspects the basic structure of a database (i.e., tables, columns, foreign and primary keys) and builds correspondences between those and the OWL elements, based on a predefined set of assumptions. Every table name is considered an equivalent to an OWL class. Columns that are not part of keys, are converted to datatype properties with corresponding *xsd* data types. Columns that are foreign keys, are converted to object properties, with appropriate range and domain. Primary keys are converted either to object properties (like foreign keys), or to is-a relations. The authors have identified a few special cases that help to increase the quality of the resulting ontology. For instance, tables that contain only two foreign keys (and no other columns) are converted to an object property, instead of a class. Tables that have columns that are both primary keys and foreign keys pointing to a primary key in another table are considered to be in a subsumption relation. If the keys are not present, the method simply does not work as intended, i.e. it only produces simple classes, with restrictions on datatype properties, and no object properties or is-a relations.

Construction of a taxonomy is the most sensitive part of this solution. Unfortunately, judging by the ontology engineering standards, the constructed taxonomy is very "flat", i.e. has very few relations. Since the meaning of data is lost in a database, any columns that serve only to identify a virtual entity in a relational database are not required in other storage types (e.g. graph and object databases have identifiers separate from other properties or fields of data objects). In the resulting ontology they have the same relevance as any other property, even though they may be deemed unnecessary by an ontology engineer (OWL entities have globally unique URIs, so other identifiers are gratuitous). N-ary relationships are also not considered specifically.

Besides the ontology, DB2OWL [15] produces a mapping between the database and the ontology in the R2O format. It contains a full description of the database schema and correspondences between the schema and the ontology. Unfortunately, according to our best knowledge, the DB2OWL has not seen any meaningful improvements since 2008, when the prototype was implemented. The software tool is not readily available online. Resulting ontologies are burdened by characteristics of relational database models. They are specific to the database, with no links to other ontologies, and have somewhat "flat" taxonomies. Following the DB2OWL algorithm, the classes necessarily need to have direct instances (database entities), and no class without instances is created, which is not in line with the OWL models.

The RDBToONTO [18], is a solution similar to the DB2OWL. It offers an open-source implementation of an automatic algorithm that constructs OWL ontologies from relational databases. A useful feature of the RDBToONTO is the responsive user interface that displays results and allows a certain degree of control over processing. Specifically, a user is able to add constraints that include table inclusion/exclusion, class and relation naming patterns, and taxonomy creation rules. Because the results of the constrained process are displayed to the user, the tool allows for a pseudo-iterative process of adjusting constraints and re-running the algorithm in order to improve the resulting ontology. The RDBToONTO is also able to populate the ontology with individuals, name patterns of which are user-defined. As a result, the OWL file is a good representation of the model and contents of the relational data. We have found that ontologies generated by the RDBToONTO are of moderate quality, which is the best result when it comes to automatic generation of OWL files from the relational databases.

Let us also mention that there are other tools, such as mapping languages (e.g. D2RQ, R2RML, RDB2OWL), and data integration tools (e.g. KARMA, Virtuoso). These kinds of software can assist in extracting explicit semantics from RDB (and other formats), but, generally, does not produce an ontology (or a EDB to OWL mapping) automatically, without user input.

D. Non-relational databases and other data formats

Let us now make few observations about data formats not considered thus far. All of them, for different reasons, "deserve" to be mentioned. Let us start from the RDF, which is a special data format in our context, because OWL uses RDF and RDFS tags directly and can be stored in the RDF format. Moreover, some key tags in RDF and RDFS are direct superclasses of OWL tags (e.g. owl:Class is a subclass of rdfs:Class, owl:Property is a subclass of rdf:Property). OWL-DL, the reasoner-enabled OWL profile, imposes some restrictions on the RDF tags that are consistent with the language definition, e.g. an entity cannot be at the same time an owl:Class and an owl:Individual. OWL Full relaxes those restrictions and can store any RDF tag, making the conversion from RDF trivial. However, because of its limited practical usability, we are not interested in OWL Full in our context. Overall, relations between RDF, RDFS, and various OWL profiles are very intricate and beyond the scope of this work.

Next, let us look at non-relational databases. In recent years, NoSQL databases have reached some level of maturity, and are utilized across many industries and applications. However, it has to be immediately observed that the NoSQL is a term that spans multiple technologies and implementations. These include document, graph, object, key-value, triplestore, multi-paradigm and other types of databases. Even within one paradigm, such as graph databases, implementations and models vary greatly between vendors. In some cases, such as triple-stores, and some graph databases, RDF is stored directly, so the problem is reduced to lifting from RDF. For the sake of completeness, let us mention that there are also NoSQL solutions that are designed for OWL specifically (e.g. Stardog).

In other cases, conversion of a database specific model, used in a given database, to OWL requires a tailor-made solution, designed for a given database engine. Even the graph model, which at a glance seems to naturally fit the OWL, is (in practice) realized through a large diversity of completely disparate implementations (both for data persistence and database queries). The existing differences in models and query languages result in lack of a general solution to "lifting" NoSQL models to OWL. As a matter of fact, literature search did not reveal any serious attempts of proposing approaches to solving this problem even for specific databases. Therefore, at present, in order to solve this problem, one is required to write scripts or programs customized to both the problem at hand and the specific database engine. As an illustration, let us consider three popular databases—MongoDB, Neo4j and OrientDB. Data formats they use are called *BSON* (JSON-like format with dynamic schemas); *Cypher graph* (with nodes, edges and attributes); and *Orient graph* (with edges, optionally typed vertices, documents, relationships and others); respectively. Those models are accessed with vendor-specific languages, such as OrientSQL, or CQL (Cypher Query Language; for Neo4j). Devising a mechanism to lift any of these models to OWL is not possible without a deep understanding of each of these models (and languages), and the differences between them might make development of a global solution impossible. Implementation of a solution for a particular database needs to use a respective query language. The implementation problem can be somewhat mitigated by the use of a language commonly supported across databases, such as the Gremlin (compatible with both Neo4j and OrientDB, but not with MongoDB).

Let us finally mention the general class of "other technologies" of data persistence and/or communication. As an example let us use software agents. Here, we see, for instance, the KQML [22] and ACL [32] communication languages and the FIPA SL Content Language (which may serve as an OWL wrapper [35] in communication). We mention these technologies specifically, because software agents naturally materialize in the context of the Internet of Things (see, for instance, [23], [25]). Also note that those are examples of domain-specific communication mechanisms, which were thought through to match the needs of the domain itself. They have been also elevated to the level of standardization (ACL and FIPA SL). Nevertheless, these communication mechanisms do not have an OWL based representation.

KQML, ACL and FIPA ACL represent a class of problems not uncommon in contemporary software. The standardized schemas are stored and represented exclusively in human-readable files, such as documents with text descriptions and diagrams. Even if the information is structured, e.g. in one of standardized UML representations, it's semantics is implicit. In other words, lifting to OWL requires manual labor and depends on understanding of the documentation. In this class of problems there are no tools whatsoever to support an ontology engineer in lifting the implicit semantics of the software communication mechanism or data model to OWL, despite complete documentation.

IV. Concluding remarks

After the analysis of methods and tools available for "lifting to OWL" we have concluded that each of the researched data formats calls for a slightly different approach.

For the XML instances and the XML Schema, there exist dedicated tools (some of them available online), however there are special XML constructs that cannot be immediately represented in OWL, and should be considered separately, e.g. by manually adding a meta-ontology.

In case of JSON, the procedure for extracting semantics differs depending on the presence or absence of the JSON Schema. If it is available, then there are tools that can convert it to the XSD, and then tools for the XSD conversion can be applied. In case of no schema, full documentation is required. Unfortunately, no tools for direct conversion of the JSON Schema to the OWL ontology are available.

For relational database schemas there exist tools for generating OWL ontologies, however the database itself may not have semantics "embedded" even in an implicit format, as it is designed with different objectives. Therefore, an iterative process involving a human may be necessary, with extra effort in case of no-semantic-carrying labels.

RDF and RDFS files are, in principle, easy to convert to OWL, but their inherent high expressivity (compared to OWL) may not translate into good ontologies.

Other data communication formats and various variants of NoSQL databases require dedicated handling since there are no tools available.

Overall, it has to be stressed that, despite the source data format, the produced ontologies need to be verified by an ontology engineer before use. In other words, automated tools cannot (yet?) replace an ontology engineer. On the other hand, the ontology engineer can find a degree of support in relevant software, and is not alone in the process of lifting to OWL. In any case, the last step in the process should always be quality control performed by a human.

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] http://ontologydesignpatterns.org/wiki/Odp:WhatIsAnExemplaryOntology.
[2] http://wiki.opensemanticframework.org/index.php/Ontology_Best_Practices.
[3] https://github.com/krg7880/json-schema-generator.
[4] http://www.altova.com/xmlspy/json-schema-editor.html.
[5] Generate schema. https://github.com/Nijikokun/generate-schema.
[6] Genson. https://github.com/wolverdude/GenSON/.
[7] JSON-LD support for OWL API. https://github.com/stain/owlapi-jsonld.
[8] JSON Schema. http://json-schema.org/.
[9] JSON Schema Net. http://jsonschema.net/.
[10] Jsons2xsd. https://github.com/ethlo/jsons2xsd.
[11] OWL API. https://github.com/owlcs/owlapi.
[12] Owl web ontology language guide. https://www.w3.org/TR/owl-guide/.
[13] Internet engineering task force (IETF). the JavaScript Object Notation (JSON) data interchange format. https://tools.ietf.org/html/rfc7159, March 2014.
[14] Mehdi Bahrami, Mokhtaria Hacherouf, and Safia Nait Bahloul. Proc. of the 2015 international conference on soft computing and software engineering (SCSE'15) DTD2OWL2: A new approach for the transformation of the DTD to OWL. *Procedia Computer Science*, 62:457–466, 2015.
[15] Jesús Barrasa, Óscar Corcho, and Asunción Gómez-pérez. R2o, an extensible and semantically based database-to-ontology mapping language. In *in In Proc. of the 2nd Workshop on Semantic Web and Databases(SWDB2004*, pages 1069–1070. Springer, 2004.
[16] I. Bedini, C. Matheus, P. F. Patel-Schneider, A. Boran, and B. Nguyen. Transforming XML schema to OWL using patterns. In *Semantic Computing (ICSC), 2011 Fifth IEEE International Conference on*, pages 102–109, Sept 2011.
[17] Hannes Bohring and Sören Auer. Mapping XML to OWL ontologies. In *Leipziger Informatik-Tage, volume 72 of LNI*, pages 147–156. GI, 2005.
[18] Farid Cerbah. Learning highly structured semantic repositories from relational databases. In *European Semantic Web Conference*, pages 777–781. Springer, 2008.
[19] Nadine Cullot, Raji Ghawi, and Kokou Yétongnon. Db2owl: A tool for automatic database-to-ontology mapping, 2007.
[20] Matthias Ferdinand, Christian Zirpins, and D. Trastour. Lifting XML Schema to OWL. In Nora Koch, Piero Fraternali, and Martin Wirsing, editors, *Web Engineering - 4th International Conference, ICWE 2004, Munich, Germany, July 26-30, 2004, Proceedings*, pages 354–358. Springer Heidelberg, 2004.
[21] Miriam Fernández, Chwhynny Overbeeke, Marta Sabou, and Enrico Motta. What makes a good ontology? a case-study in fine-grained knowledge reuse. In *Asian Semantic Web Conference*, pages 61–75. Springer, 2009.
[22] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. Kqml as an agent communication language. In *Proc. of the 3rd International Conference on Information and Knowledge Management*, pages 456–463. ACM, 1994.
[23] Giancarlo Fortino, Antonio Guerrieri, Michelangelo Lacopo, Matteo Lucia, and Wilma Russo. *An Agent-Based Middleware for Cooperating Smart Objects*, pages 387–398. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
[24] Giancarlo Fortino, Antonio Guerrieri, Wilma Russo, and Claudio Savaglio. *Middlewares for Smart Objects and Smart Environments: Overview and Comparison*, pages 1–27. Springer International Publishing, Cham, 2014.
[25] Giancarlo Fortino, Antonio Guerrieri, Wilma Russo, and Claudio Savaglio. *Middlewares for Smart Objects and Smart Environments: Overview and Comparison*, pages 1–27. Springer International Publishing, Cham, 2014.
[26] Michael Friedewald and Oliver Raabe. Ubiquitous computing: An overview of technology impacts. *Telematics and Informatics*, 28(2):55–65, 2011.
[27] Francis Galiegue, K Zyp, and Gary Court. Internet engineering task force (IETF). JSON Schema: core definitions and terminology. https://tools.ietf.org/html/draft-zyp-json-schema-04, August 2013.
[28] Maria Ganzha, Marcin Paprzycki, Wiesław Pawłowski, Paweł Szmeja, and Katarzyna Wasielewska. *Towards semantic interoperability between Internet of Things platforms*. Springer, submitted for publication, 2016.
[29] Maria Ganzha, Marcin Paprzycki, Wiesław Pawłowski, Paweł Szmeja, Katarzyna Wasielewska, and Giancarlo Fortino. Tools for ontology matching—practical considerations from INTER-IoT perspective. In *Proc. of the 8th Int. Conference on Internet and Distributed Computing Systems*, volume 9864 of *LNCS*, pages 296–307. Springer, 2016.
[30] Raji Ghawi and Nadine Cullot. Building ontologies from XML data sources. In *Proc. of the 20th International Workshop on Database and Expert Systems Application*, DEXA '09, pages 480–484, Washington, DC, USA, 2009. IEEE Computer Society.
[31] Thomas R. Kramer, Benjamin H. Marks, Craig I. Schlenoff, Stephen B. Balakirsky, Zeid Kootbally, and Anthony Pietromartire. Software tools for XML to OWL translation. Technical report, NIST Interagency/Internal Report (NISTIR) – NIST IR 8068, July 2015.
[32] Yannis Labrou, Tim Finin, and Yun Peng. Agent communication languages: The current landscape. *IEEE Intelligent systems*, 14(2):45–52, 1999.
[33] Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoc. Foundations of JSON schema. In *Proc. of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 263–273, 2016.
[34] Toni Rodrigues, Pedro Rosa, and Jorge Cardoso. Mapping XML to Existing OWL Ontologies. In Pedro Isaías, Miguel B. Nunes, and Inmaculada J. Martínez, editors, *International Conference WWW/Internet 2006*, pages 72–77, 2006.
[35] Bernhard Schiemann and Ulf Schreiber. Owl dl as a fipa acl content language. In *Proc. of the Workshop on Formal Ontology for Communicating Agents (FOCA), 18th European Summer School of Language, Logic and Information*, pages 73–80, 2006.
[36] Chrisa Tsinaraki and Stavros Christodoulakis. *Interoperability of XML Schema Applications with OWL Domain Knowledge and Semantic Web Tools*, pages 850–869. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.
[37] Chrisa Tsinaraki and Stavros Christodoulakis. *XS2OWL: A Formal Model and a System for Enabling XML Schema Applications to Interoperate with OWL-DL Domain Knowledge and Semantic Web Tools*, pages 124–136. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.
[38] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, January 1991.
[39] Martin Wischenbart, Stefan Mitsch, Elisabeth Kapsammer, Angelika Kusel, Stephan Lechner, Birgit Pröll, Werner Retschitzegger, Johannes Schönböck, Wieland Schwinger, and Manuel Wimmer. Automatic data transformation: Breaching the walled gardens of social network platforms. In *Proc. of the APCCM'13*, pages 89–98. Australian Computer Society, Inc., 2013.
[40] Nora Yahia, Sahar A. Mokhtar, and AbdelWahab Ahmed. Automatic generation of OWL ontology from XML data source. *CoRR*, abs/1206.0570, 2012.