This is the author's version of an article published in Proceedings of the 45th Annual Conference of the IEEE Industrial Electronics Society (IECON 2019).
Changes were made to this version by the publisher prior to publication. The final version is available at: https://doi.org/10.1109/IECON.2019.8927005

Annotation Rules for XML Schemas with Grouped Semantic Annotations

Rogerio Campos-Rebelo* † ‡, Filipe Moutinho* †, Luís Paiva*, and Pedro Maló*
*NOVA University Lisbon, NOVA School of Science and Technology, Monte de Caparica, Portugal
†UNINOVA, Center of Technology and Systems, Monte de Caparica, Portugal
‡Polytechnic Institute of Beja, School of Technology and Management, Beja, Portugal
rcr@uninova.pt, fcm@fct.unl.pt, luismpaiva@uninova.pt, pmm@uninova.pt

***Abstract*—To enable the Industrial Internet of Things (IIoT), it is required to ensure Machine-to-Machine communications. Systems/devices often use different communication protocols, standards, and data representation languages, which create interoperability challenges. This paper proposes a set of annotation rules for systems meta-data, to support the translation of data exchanged between heterogeneous systems. These rules must be followed to ensure the validity of systems meta-data (XML Schemas annotated with semantic annotations and group identifiers). The meta-data can then be used as input in tools to analyze data and semantic compatibility and generate translators.**

***Index Terms*—Data Translation, Interoperability, Internet of Things (IoT), Semantic Annotation**

I. Introduction

The Internet-of-Things (IoT) applied to the industry creates not only many opportunities but also a lot of challenges [1]. The Industrial IoT (IIoT) must support the interaction between several industrial systems/machines, namely sensors, controllers, and actuators. The existence of multiple standards, protocols, syntaxes, and semantics, present a set of interoperability challenges, when it is intended that, heterogeneous industrial systems, from different sources, communicate with each other.

There are three levels of interoperability challenges, created by the existence of different communication protocols (HTTP, CoAP, MQTT, ...), data formats, and semantics. These are often referred to as technical interoperability, syntactic interoperability and semantic interoperability [2]. The technical interoperability will not be addressed in this paper. Regarding the data format, systems usually exchange messages in XML (Extensible Markup Language) or JSON (JavaScript Object Notation). The interaction between systems that use these two data formats can be easily supported by a data format translator.

This paper addresses the semantic interoperability, defined in [3] as a shared understanding between the exchanged data/information. When two systems cannot understand each other messages (their data semantics), translators are required. These type of systems will be named to throughout the paper as heterogeneous systems. Considering the huge number of heterogeneous systems, each one sending or receiving messages with specific data and semantics, it is not possible to create/provide translators for each pair of heterogeneous systems. This task, should not be done by humans. Highly heterogeneous systems should interoperate dynamically [2].

To dynamically support the semantic interoperability between heterogeneous systems, an approach was proposed in [4]. In this approach, it is verified if two systems can interact and, if possible, a translator is automatically generated. To do it, systems meta-data (XML Schemas) with semantic annotations must be available. These annotations, not only map the XML elements/attributes of the exchanged messages to the concepts/properties of a reference ontology, but also group them to solve ambiguities.

This paper proposes a set of annotation rules to ensure proper/valid semantic annotations in the XML Schemas (XSD) that support the approach proposed in [4]. It was defined, in [4], how to create each semantic annotation; however, it was not defined how to ensure that the set of annotations in an XSD are valid, i.e. that annotations not then in "conflict" with each other.

The approach proposed in [4] provides a contribution to the Arrowhead framework [5], [6]. The Arrowhead software framework is a service-oriented architecture (SOA) framework offering a set of services to support the interaction between application systems (service providers and service consumers). To support the translation of the data exchanged between those application systems, if heterogenous, the approach proposed in [4] may be used. Using this approach, from annotated XSDs and reference ontologies, it is possible to verify if heterogeneous systems are semantically compatible, and if so, translators can be automatically generated.

This paper is divided in six sections. In next one, the Arrowhead Framework is presented. Then, in section III, the translation approach, proposed in [4], [7], is presented. Semantic annotations for XML Schemas are described in section IV. The proposed annotation rules are presented in section V. Finally, conclusions are presented the VI.

II. Arrowhead Framework

The Arrowhead software framework [6] is a service-oriented architecture (SOA) framework. It was initially developed in the Arrowhead project [6] and, currently, it is being developed in the Productive4.0 project [8] and in Arrowhead-Tools project [9]. This software framework supports the interaction between systems. To enable such interaction, a specific consumer system must be able to access the information provided by a provider system. For this, it is necessary, in the first instance, to discover a provider that offers the service desired by the consumer. It is still necessary to find out if the consumer can communicate with the provider that offers the service desired by him. The main services provided by the Arrowhead framework and presented in Fig. 1, are:

- Service registry (SR) – which allows a provider to register a service that will become available;
- Service discovery (SD) – which allows a client to search for a service among the available services;
- Orchestration (ORC) – which informs the consumers about the most suited providers for them;
- Autorization (AUT) – which allows the provider to be sure that the consumer can consume the required service.

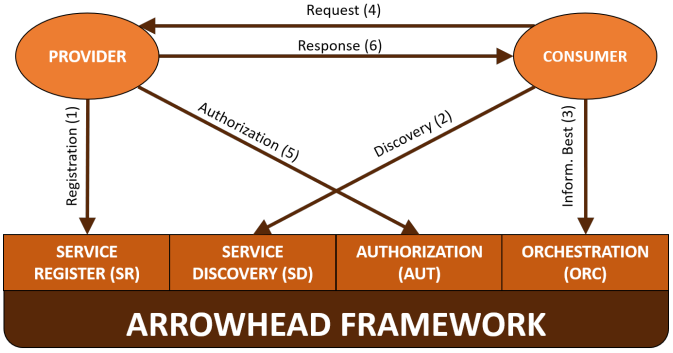

Fig. 1. Interaction of a provider, a consumer, and the Arrowhead framework.

The way how a consumer and a provider can interact, with the Arrowhead software framework, is presented in Fig. 1. For a service to be available, initially a provider must communicate with Service Registry to register a service (1). From that moment on, the Arrowhead framework is aware that this provider offers this particular service. The consumer begins its interaction with the Arrowhead framework, through the Service Discover (2), asking if there are providers that offer the desired service. If the service is available, the consumer can communicate with the orchestration (3), which analyzes the providers that offer the service and, based on consumer meta-data, returns the information about the best provider to offer the desired service. After this, the consumer can communicate directly with the provider. The Consumer sends a request message to the Provider (4) asking for the desired service. When the request is received, the provider may send a message to the Authorization asking if it can communicate with that consumer (5). If the answer is positive, the Provider responds to the request by sending a message to the consumer with the data related to the requested service (6).

III. Translation Approach

To enable the communication between a provider and a consumer, it is not enough for the provider to offer the service that the consumer wants. It is also necessary that both ”speak the same language”, that is, the consumer must be able to understand the data made available by the provider. This constraint limits the choice in the orchestration, and may lead that, although there are several providers registered to offer the service desired by the consumer, none of them is authorized to offer it to that consumer. To deal with this problem, translators are required, translating the exchanged messages into the receiver ”language” (with its syntax and semantics).

Fig. 2 extends the Fig. 1 with a translator. This translates the messages exchanged between the provider and the consumer, enabling their communication after their interaction with the Arrowhead framework. In this scenario, messages are sent to the translator, which translates the messages, generating new messages and sending them to the receiver. The translator can be third part system or can be embedded in the sender or in the receiver system.

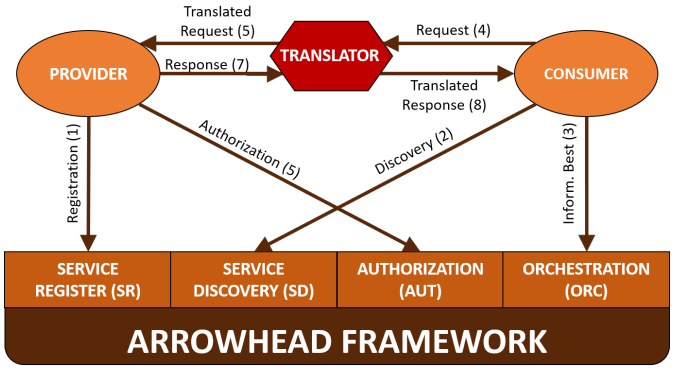

Fig. 2. Interaction of a provider, a consumer, and the Arrowhead framework, supported by a translator.

The translators can be manually developed, but to do it, it is necessary to know a priori all the providers and consumers that can communicate, to develop a translator for each pair. This means that, if even possible, developing the translators manually would imply a considerable effort.

To make the translation process more flexible, a tool prototype, to automatically generate translators, was developed in [4], [7]. This tool uses the Producer and Consumer meta-data to verify if they can communicate, and if so, generate the translator. Systems meta-data must be XSD files. These XML Schemas must have semantic annotations based on a reference ontology in Web Ontology Language (OWL) [10] format.

For each pair of heterogeneous systems, the tool [4], [7] receives three input files and, if possible, generates one output file, the translator. The input files are the reference ontology and two XSDs, one for the provider system and the other for the consumer system. The tool verifies if the XSDs are semantically compatible, and if so, generates the translator, an XSLT (eXtensible Stylesheet Language Transformation) [11] file. This XSLT translates sender XML messages into receiver XML messages. This tool prototype is available online at *http://gres.uninova.pt/tag/*.

IV. Semantic Annotations

The mentioned semantic annotations, which provide meaning to data messages exchanged between heterogeneous systems, are described in this section. These semantic annotations are inserted into the XSD files, the meta-data, that describe the exchanged XML messages.

A. SAWSDL

These annotations are made using Semantic Annotations for Web Service Description Language (SAWSDL) [12], which is the W3C recommendation for annotating XML Schemas. With SAWSDL, each XSD element can be annotated with ontology concepts. An example of a semantic annotation using SAWSDL is presented in (1). In this example the XML element "indoorTemp" is annotated with the concept "IndoorTemperature" from the ontology presented in Fig. 3.

$$<xs:element\ type="xs:float"\ name="indoorTemp"\ sawsdl:modelReference="/IndoorTemperature"/> \quad (1)$$

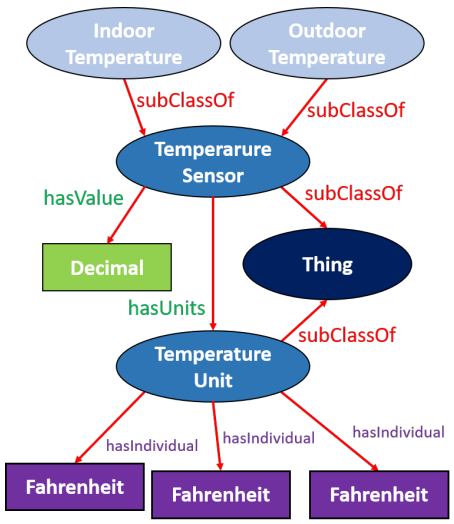

Fig. 3. Example of an ontology for temperature sensors

B. Annotation Path

The annotation paths, proposed in [13] [14], enable the creation of more expressive annotations. For instance, it is possible to annotate XSD elements with ontology concepts and properties, not only with concepts, as in SAWSDL. The elements of an XSD Schema are annotated with annotation paths, where each path is a sequence of steps and each step is a concept or a property of the reference ontology. In an annotation path, the odd steps are always ontology concepts and the event steps are always ontology properties (object property or data type property). An object property cannot be the final step; whereas a data type property cannot be a middle step. Additionally, concept steps may have restrictions. The XSD element annotated in (1) using SAWSDL is annotated in (2) with an annotation path. In (2), the first step (an even step) is a concept step and the second step (an odd step and the final one) is a data type property. Another example of an annotation path is presented in (3). In this example, the first and the third steps are concepts and the second is an object property.

$$<xs:element\ type="xs:float"\ name="indoorTemp"\ sawsdl:modelReference="/IndoorTemperature/hasValue"/> \quad (2)$$

$$<xs:element\ type="xs:string"\ name="unitsa"\ sawsdl:modelReference="/TemperatureSensor/hasUnits/TemperatureUnits"/> \quad (3)$$

An example of an XML message is presented in Fig. 4 and the associated XSD with semantic annotations, using annotation paths, is presented if Fig. 5. The XML elements "sensorTemp1", "unitsa", "sensorTemp2", and "unitsb", contain values and units of temperature sensors. Each of these elements has an associated semantic annotation, as presented in Fig. 5. "sensorTempX" has the semantic annotation presented in (2); whereas the "unitsY" has the semantic annotation presented in (3).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <data>
3     <sensorTemp1>72.3</sensorTemp1>
4     <unitsa>F</unitsa>
5 <sensorTemp2>25.3</sensorTemp2>
6     <unitsb>C</unitsb>
7 </data>
```

Fig. 4. Example of an XML message

C. APP Info

To support the translation of data values, the introduction of data values, mapped into ontology individuals, was proposed

```
<?xml version="1.0" encoding="UTF-8"?>
…
  <xs:complexType>
   <xs:sequence>
    <xs:element type="xs:float" name="sensorTemp1"
   sawsdl:modelReference="/TemperatureSensor/hasValue"/>
    <xs:element type="xs:string" name="unitsa"
     sawsdl:modelReference="/TemperatureSensor/
    hasUnits/TemperatureUnits" />
 <xs:element type="xs:float" name="sensorTemp2"
   sawsdl:modelReference="/TemperatureSensor/hasValue"/>
    <xs:element type="xs:string" name="unitsb"
     sawsdl:modelReference="/TemperatureSensor/
    hasUnits/TemperatureUnits" />
  …
```

Fig. 5. XSD of the XML from Fig. 4 with annotation paths referring the ontology from Fig. 3.

in [7]. To explain it, the XML message sent by a provider (Fig. 6) will be used. This provider sends the values ”Fah” and ”Cel” to indicate which are the temperature units (fahrenheit or celsius) of the temperature values it sends. Supposing that a consumer is expecting to receive an XML, such as the one presented in Fig. 4, it understands ”F” and ”C” but not ”Fah” and ”Cel”. To deal with this type of problem, another type annotations are appended to the XSD, in the $xs{:}appinfo$ of the $xs{:}annotation$, as presented in Fig. 7. Additionally, it is required to insert the mapping references (”a3st:mpi-ref”) into the XSD element, as presented in line 5 of Fig. 7. Fig. 7 presents the APP Info and the mapping references that must be added into the Consumer XML Schema (Fig. 5). In this example, the ”C”, ”F” and ”K” units, are associated with the individuals ”Celsius”, ”Fahrenheit” and ”Kelvin” of the ontology ”Temperature Unit” concept.

```
<?xml version="1.0" encoding="utf-8" ?>
<values>
    <indoorTemp>72.3</indoorTemp>
    <units1>Fah</units1>
<outoorTemp>25.3</outdoorTemp>
    <units2>Cel</units2>
</values>
```

Fig. 6. Example of a Provider XML

```
<?xml version="1.0" encoding="UTF-8"?>
…
    <xs:element type="xs:string" name="units"
      sawsdl:modelReference="/TemperatureSensor/
      hasUnits/TemperatureUnits" a3st:mpi-ref="1;2;3"/>
    ...
    <xs:annotation>
       <xs:appinfo>
        <a3st:map-data-ind a3st:individual="Celsius"
          a3st:mdi-id=”1”>C</a3st:map-data-ind>
        <a3st:map-data-ind individual="Fahrenheit"
          a3st:mdi-id=”2”>F</a3st:map-data-ind>
        <a3st:map-data-ind a3st:individual="Kelvin"
          a3st:mdi-id=”3”>K</a3st:map-data-ind>
       </xs:appinfo>
    </xs:annotation>
</xs:schema>
```

Fig. 7. Example of an annotated XML Schema with APPinfo

D. Grouping Semantic Annotations

Annotation paths with groups were proposed in [4] to group/associate XML elements/attributes. Provider systems and consumer systems, can send or expect, in a single message:

- data from multiple things - such as in the XML presented in Fig. 4, which includes data from two temperature sensors (”1” and ”2”);
- multiple data of one thing - for instance, the XML presented in Fig. 4 contains, for each temperature sensor, its temperature value and units.

Path annotations, presented in [13] [14], provide meaning to each XML element or attribute; however, they are not enough to support the creation of groups of XSD elements. It is not possible associate the ”sensorTemp1” element with the ”unitsa” element nor the ”sensorTemp2” element with the ”unitsb” element.

An annotation path with groups is an annotation path were each concept step can have a set of group IDs. Group IDs are only valid within its XSD. This means that group IDs in provider and consumer XSDs are independent.

To create groups of XSD elements (XML elements/attributes) the extension proposed in [4] must be used. For example, if ”sensorTemp1” and ”unitsa”, from Fig. 4, provide meaning about the same temperature sensor (”1”) and ”sensorTemp2” and ”unitsb”, from the same XML message, provide meaning about another temperature sensor (”2”), then it must be specified in the XSD, to avoid ambiguities and translation mismatches. The annotation paths presented in (4) and (5) are the same annotations presented in (2) and (3), but with groups. The XSD of the XML from Fig. 4 with annotation paths with groups, is presented in Fig. 8.

$$<xs{:}element\ type{=}”xs{:}float”\ name{=}\ ”indoorTemp”\ sawsdl{:}modelReference{=}\ ”/IndoorTemperature\{1\}/hasValue”/> \quad (4)$$

$$<xs{:}element\ type{=}”xs{:}string”\ name{=}”unitsa”\ sawsdl{:}modelReference{=}/”Temperatu\ reSensor\{1\}/hasUnits/TemperatureUnits”/> \quad (5)$$

V. Annotation Rules

The proposed annotation rules, for XML Schemas with annotation paths with groups, are presented in this section. In [4], it was defined how to create a valid annotation path with group IDs; however, it was not defined how to ensure that the set of annotation paths, with groups, in an XSD, are valid.

To simplify the proposed rules presentation, the ontology presented in Fig. 9 will be used. This short ontology was

```
<?xml version="1.0" encoding="UTF-8"?>
…
 <xs:complexType>
  <xs:sequence>
   <xs:element type="xs:float" name="sensorTemp1"
  sawsdl:modelReference="/TemperatureSensor{1}/hasValue"/>
   <xs:element type="xs:string" name="unitsa"
    sawsdl:modelReference="/TemperatureSensor{1}/
   hasUnits/TemperatureUnits" />
<xs:element type="xs:float" name="sensorTemp2"
  sawsdl:modelReference="/TemperatureSensor{2}/hasValue"/>
   <xs:element type="xs:string" name="unitsb"
    sawsdl:modelReference="/TemperatureSensor{2}/
   hasUnits/TemperatureUnits" />
…
```

Fig. 8. XSD of the XML from Fig. 4 with annotation paths with groups.

created for this purpose only. It contains 7 concepts: ”Home” and ”Room” are sub classes of ”Thing”; ”House” and ”Apartment” are sub classes of ”Home”; and ”Bedroom” and ”DiningRoom” are sub classes of ”Room”. Homes can have rooms; whereas rooms can have area value and height value.

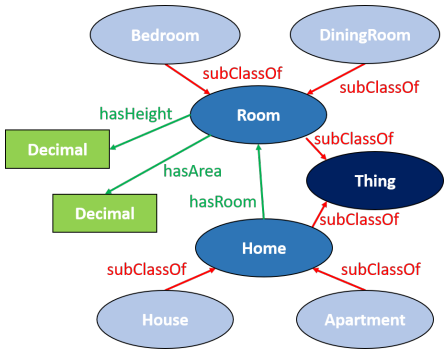

Fig. 9. Example of an ontology for Home definition

The following rules only apply to annotation paths with groups that belong to the same XSD. They ensure proper/valid semantic annotations in XSDs. In the following rules definition, when it is said that two concepts are semantically related, it means that:

- they are the same concept;
- they are equivalent classes (*equivalentClass*); or
- one is a sub class of the other (*subClassOf*).

***RULE 1**: If two concepts (of two different annotation paths):*

- *are semantically related and*
- *have the same group ID,*

then they have to be at the same position/step in their annotation paths. To explain it, two examples are presented:

- Example 1: An XSD just with the annotations (6) and (7) is correctly annotated because, in both annotations:
  - the ”Home” concepts (the same concept) with ID ”1” (the same group ID) are in the same position/step (first position);
  - the ”DiningRoom” and ”Room” concepts (one is a sub class of the other), with ID ”7” (the same group ID), are in the same position/step (third position).

  From these two annotation paths, it is possible to conclude that:
  - the ”Room” from annotation (7) is a ”DiningRoom”; and
  - the XML elements/attributes annotated in the XSD with these annotation paths, provide the area and height of the dinning room with the ID ”7”, of the home with the ID ”1”.
- Example 2: An XSD with annotations (7) and (8) is not correctly annotated because:
  - ”Bedroom” concept, from (8), is semantically related (*subClassOf*) to ”Room” from (7) and have the same ID, but are not in the same position/step (”Bedroom” is at the first position and ”Room” is at the third position).

  These two annotation paths are ambiguous. It is not possible to conclude if the bedroom with the ID ”7” is the room with ID 7 from the house with ID ”1” (annotation (7)).

$$Home\{1\}/hasRoom/DiningRoom\{7\}/hasArea \quad (6)$$

$$Home\{1\}/hasRoom/Room\{7\}/hasHeight \quad (7)$$

$$Bedroom\{7\}/hasHeight \quad (8)$$

***RULE 2**: If two concepts (of two different annotation paths):*

- *are sub class of the same concept and*
- *all their previous steps are semantically related and have the same group IDs,*

then they cannot have the same group ID. To clarify it, a set examples are presented:

- Example 3: An XSD just with annotations (6) and (9) is correctly annotated because although:
  - they (”DiningRoom” and ”Bedroom”) are sub class of the same concept (”Room”) and
  - all their previous steps are semantically related and have the same group IDs,

  their group IDs are different (”7” and ”8”). From these two annotation paths, it is possible to conclude that:
  - the ”Home” from annotation (6) is a ”House”;
  - the XML element/attribute annotated in the XSD with annotation (6), provide the area of the dinning room with the ID ”7”, of the house with the ID ”1”; and
  - the XML element/attribute annotated in the XSD with annotation (9), provide the height of the bedroom with the ID ”8”, of the house with the ID ”1”.

- Example 4: An XSD just with annotations (6) and (8) is correctly annotated because although they (”DiningRoom” and ”Bedroom”):
  - are sub class of the same concept (”Room”) and
  - and have the same group ID (”7”),

  their previous steps are not semantically related.
- Example 5: An XSD just with annotations (9) and (10) is correctly annotated because although they (”Bedroom” and ”DiningRoom”):
  - are sub class of the same concept (”Room”);
  - and have the same group ID (”8”); and
  - their previous steps are semantically related;

  their previous steps do not have the same group IDs (”1” and ”empty”). From these two annotation paths, it is possible to conclude that:
  - the XML element/attribute annotated in the XSD with annotation (9), provide the height of the bedroom with the ID ”8”, of the house with the ID ”1”; and
  - the XML element/attribute annotated in the XSD with annotation (10), provide the height of the dining room with the ID ”8”, of the home without (with default) group ID.

$$House\{1\}/hasRoom/Bedroom\{8\}/hasHeight \quad (9)$$

$$Home/hasRoom/DiningRoom\{8\}/hasHeight \quad (10)$$

RULE 3*: If there are two related concepts, of two different annotation paths, without group ID, then they have a default group ID, which is different from all defined group IDs of their related concepts.* This is why, in the example 5, ”House” concept from annotation (9) had not the same group ID of ”Home” concept from annotation (10). It is important to note that, if the group IDs from annotations (6) and (7), discussed in example 1, were removed, their XSL would also be well annotated.

VI. Conclusions

To produce correctly annotated XML Schemas, the developer must take into account a set of rules to ensure proper relations between semantic annotations of the same XML schema. These XSD files together with a reference ontology, can then be used as inputs in the tool [4] to generate the translators that support the interaction between provider systems and consumer systems.

As future work, it is planned to support JSON and develop a tool, to validate each XML Schema. In order to allow the translation of JSON messages, XSLT 3.0 and JSLT will be used. The new tool will verify if each XSD follows the proposed rules. This tool will then be integrated with the tool [4].

Acknowledgment

Research leading to these results has received funding from: EU ECSEL Joint Undertaking (JU), under grant agreement n° 826452 (project Arrowhead Tools); EU ARTEMIS and ECSEL JU fundings, under grant 737459 (project Productive4.0); and partners national funding authority FCT under the framework of project UID/EEA/00066/2019.

References

[1] L. D. Xu, W. He, and S. Li, “Internet of things in industries: A survey,” *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, Nov 2014.

[2] IERC. Internet of Things IoT Semantic Interoperability: Research Challenges, Best Practices, Recommendations and Next Steps EUROPEAN RESEARCH CLUSTER ON THE INTERNET OF THINGS. Accessed on 2019-04-30. [Online]. Available: http://www.internet-of-things-research.eu/pdf/IERC_Position_Paper_IoT_Semantic_Interoperability_Final.pdf

[3] P. M. N. Maló, “Hub-and-spoke Interoperability: an out of the skies approach for large-scale data interoperability,” Ph.D. dissertation, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2013.

[4] F. Moutinho, L. Paiva, J. Köpke, and P. Maló, “Extended semantic annotations for generating translators in the arrowhead framework,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2760–2769, June 2018.

[5] F. Blomstedt, L. L. Ferreira, M. Klisics, C. Chrysoulas, I. M. de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, and P. Varga, “The arrowhead approach for SOA application development and documentation,” in *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, Oct 2014, pp. 2631–2637.

[6] Arrowhead — Ahead of the future. Accessed on 2019-05-14. [Online]. Available: https://www.arrowhead.eu/

[7] F. Moutinho, L. Paiva, P. Maló, and L. Gomes, “Semantic annotation of data in schemas to support data translations,” in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2016, pp. 5283–5288.

[8] Productive 4.0 - A European co-funded innovation and lighthouse project on Digital Industry. Accessed on 2019-05-14. [Online]. Available: https://productive40.eu//

[9] Arrowhead-tools. Accessed on 2019-05-30. [Online]. Available: https://arrowhead.eu/arrowheadtools

[10] OWL 2 Web Ontology Language: Document Overview. OWL Working Group, World Wide Web Consortium Recommendation. Accessed on 2019-05-08. [Online]. Available: https://www.w3.org/TR/owl2-overview/

[11] XSL Transformations (XSLT) Version 2.0. Accessed on 2019-05-14. [Online]. Available: https://www.w3.org/TR/xslt20/

[12] Semantic Annotations for WSDL and XML Schema. Accessed on 2019-05-14. [Online]. Available: https://www.w3.org/TR/sawsdl/

[13] J. Köpke and J. Eder, “Semantic annotation of xml-schema for document transformations,” in *On the Move to Meaningful Internet Systems: OTM 2010 Workshops*, R. Meersman, T. Dillon, and P. Herrero, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 219–228.

[14] J. Köpke, “Annotation paths for matching xml-schemas,” *Data Knowledge Engineering*, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169023X17300654