# OntoEDIFACT: An Ontology for the UN/EDIFACT Standard

Boulares Ouchenne and Mhamed Itmi

Normandie Univ, INSA Rouen, LITIS, 76000 Rouen, France
Email: {boulares.ouchenne,mhamed.itmi}@insa-rouen.fr

***Abstract*—In this paper, we propose an OWL encoded ontology (OntoEDIFACT) to ontologize the United Nations/Electronic Data Interchange For Administration, Commerce and Transport (UN/EDIFACT) standard. Our ontology provides a lightweight representation that captures general concepts about basic components of the standard, and also provides extensibility for adding complex components in a hierarchical manner. Our approach separates the conceptualization from knowledge base (KB) integration. So, we start by the conception of the knowledge model and we finish by building the KB through instantiating the knowledge model from the last version of the standard. The knowledge base consists of a triple stores, publicly available through a SPARQL endpoint. We have developed some services for exploiting the KB and discussed some future applications.**

***Keywords–Ontology; EDI; EDIFACT; SPARQL-Endpoint.***

## I. INTRODUCTION

Generally, E-commerce is associated with buying and selling operations that are carried out via the Internet. This is a very biased view because E-commerce includes any transaction in which, the parties interact electronically. Electronic Data Interchange (EDI) [1][2][3] enables the exchange of structured business documents (purchase orders, invoices, etc.) between IT systems of trading partners. The use of a structured and readable format allows transferring of documents from one application to another located in a different locations, without any human interpretation and/or intervention. The EDI was designed to replace the transmission of information through paper and to overcome inefficient manual document exchange. Basically, EDI is designed with respect to the principle that the data should be entered once into the system, then it can be transmitted electronically among interested parties. In the most common scenario, the cycle starts when a buyer sends an EDI purchase order to a seller. The latter, first sends an acknowledgement to the buyer, then at the time of shipment, he sends a shipping notice followed by an invoice. All these documents are transmitted through EDI. Finally, the buyer sends his bank account information for the payment of the invoice, and funds are electronically transferred to the seller's bank account.

The design of EDI seems to be simple, but its implementation requires a thorough consensus on data elements, codes, rules of syntax and format. The exchange of electronic information is built on a common, universal, multi-sector language allowing an open and easy communication between all economic stakeholders. In other words, we can transfer data between heterogeneous systems to the extent that we use a common format. There are various standards that constitute the basis for a specific-area data exchange. A few examples are NIEM, AINSI X12, EDIFACT and XBRL. Each standard is characterized by its scope of use (North of America for AINSI X12 and NIEM, EDIFACT for the international).

UN/EDIFACT is the international EDI standard developed by the United Nations. This standard specifies a set of international standards, directories and guidelines for the electronic interchange of structured data. UN/EDIFACT provides a set of data structures (called MESSAGES) each of which, serves to transmit a particular message (Purchase order, Invoice, etc.). Each of these structures is an aggregate of content items (SEGMENTS and ELEMENTS). UN/EDIFACT does not define the medium by which the message is sent, or the protocols used in any particular form of communication. The standard is completely neutral in this aspect. It focuses only on the content of messages.

The UN/EDIFACT is a standard for EDI trading widely recognized by both commercial and non-commercial sectors. Recently, organizations have a tendency to adopt UN/EDIFACT to the long term for a structured governance with an international visibility. An example (taken from Wikipedia [4]) of an EDIFACT message used to answer to a flight ticket (FRA-JFK-MIA) availability request is presented below:

```
UNA:+.? '
UNB+IATB:1+6XPPC+LHPPC+940101:0950+1'
UNH+1+PAORES:93:1:IA'
MSG+1:45'
IFT+3+XYZCOMPANY AVAILABILITY'
ERC+A7V:1:AMD'
IFT+3+NO MORE FLIGHTS'
ODI'
TVL+240493:1000::1220+FRA+JFK+DL+400+C'
PDI++C:3+Y::3+F::1'
APD+74C:0:::6++++++6X'
TVL+240493:1740::2030+JFK+MIA+DL+081+C'
PDI++C:4'
APD+EM2:0:1630::6+++++++DA'
UNT+13+1'
UNZ+1+1'
```

Beside the widespread adoption of the UN/EDIFACT, the standard suffers from its poor design, confusing or a lack of semantics and its complicated formatted text which is non-understandable for a non-specialist. Those difficulties pushed us to propose an ontology to unambiguously specify the meaning of components of the UN/EDIFACT standard and relationships among them.

This paper is organized as follows. Section 2 provides a detailed description of the OntoEDIFACT ontology. Section 3 presents the software architecture of our prototype, freely

 ISBN: 978-1-61208-558-6 

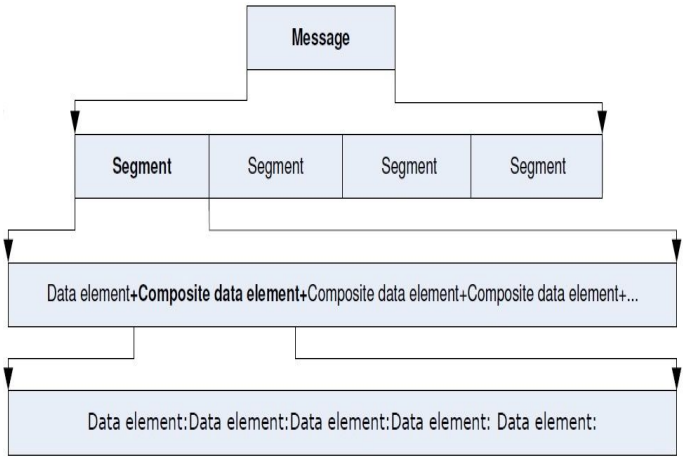


Figure 1. EDIFACT messages structure.

accessible via the Web. Section 4 briefly discusses some related works. Finally, we give our conclusions and outline future works in Section 5.

## II. Designing the OntoEDIFACT Ontology

This section describes the approach used to design the OntoEDIFACT ontology, as well as the four main components of OntoEDIFACT, namely simple elements, composite elements, segments and messages. We have defined several requirements to which our ontology must answer. First, we want to develop a general ontology in order to (i) be apprehended by the EDI community without the need to be an expert of UN/EDIFACT standard, (ii) to be independent of the version of the UN/EDIFACT standard and (iii) the structure of our ontology must also facilitate its settlement and its evolution by using possibilities of expression offered by the description languages (in terms of knowledge representation and reasoning). During the design of our ontology, we have endeavored to apply the commonly recommended techniques of the community [5][6]. Finally, since the OntoEDIFACT is described in OWL2 [7], we have taken advantage of possibilities offered by this language in terms of expressiveness.

### A. The structure of EDIFACT Messages

A in EDIFACT format is structured as depicted in Figure 1. A message is composed of an ordered set of segments. Segments can be grouped in groups which comprises an ordered set of segments. Furthermore, the message structure defines whether data segments and segment groups are mandatory or optional, and indicates how many times a particular segment or a group can be repeated. A segment comprises an ordered list of stand-alone data elements and/or composite data elements. The segment definition indicates the data elements to be included in the segment, the sequence of the data elements and whether each data element is mandatory or optional. A composite data element comprises an ordered list of two or more component data elements. The composite data element definition specifies the component data elements to be included in the composite data element, the sequence of the component data elements and whether each component data element is mandatory or optional.

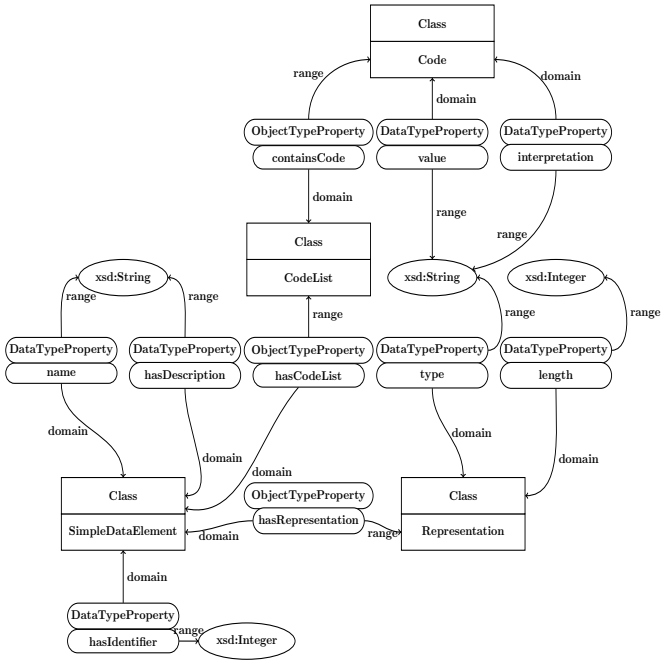


Figure 2. Simple Element Modeling.

### B. Modeling of Simple Data Elements

Simple Data Elements are the primary unit which, correspond to the lowest level of data in EDIFACT standard. They represent data types and contain single data element value. A simple data element is defined by:

- Its EDIFACT identification code number.
- Its name and its description.
- Its structure that defines its type and length.
- Eventually a predefined code list that it can take.

Figure 2 shows the OWL serialization of simple data elements. The model is structured around a set of abstract entities, each describing a conceptual object (Code, Representation, etc.). Each entity is associated with its attributes (represented by `owl:DatatypeProperty`) and relations with other entities (represented by `owl:ObjectProperty`).
A simple data element is modeled by an OWL class `SimpleDataElement`. A set of data properties have been introduced to express the relationship between objects and their data values, such as numerical values (i.e., `hasIdentifier` to represent the identification) and textual values (i.e., `hasDescription` for the description and `name` for the name of the simple data element). We have also introduced two data object properties (`hasRepresentation` and `hasCodeList`) to express respectively, the relationship between a simple data element and its representation or its code list. There are two kinds of simple data elements:

1) Simple data elements with free values (e.g., the simple data element account name[1]): in order to model valid formats of data elements values, an OWL class `Representation` and two data properties

[1] http://www.unece.org/fileadmin/DAM/trade/untdid/d16b/tred/tred1146.htm

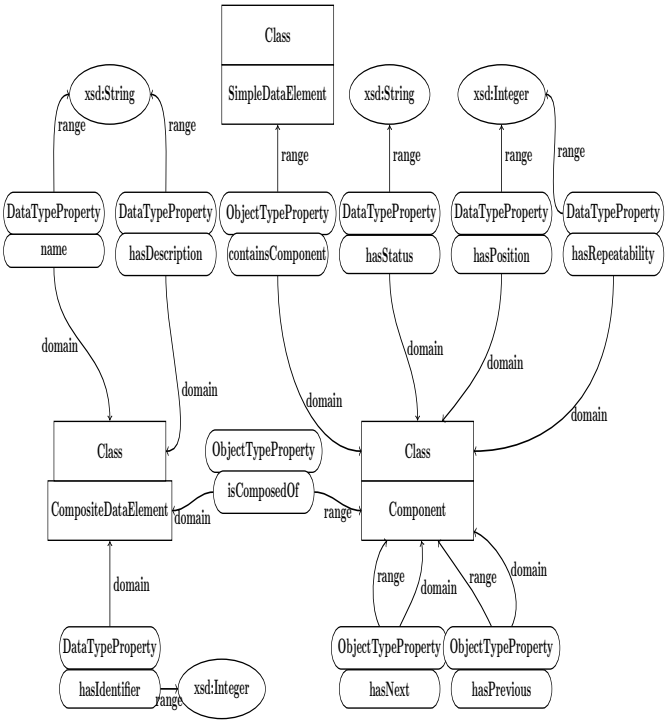


Figure 3. Composite Element Modeling.

are introduced. The first one (`type`), is used to define whether the characters used for representing the simple data element are numeric, alphabetic or alphanumeric. The second one (`length`) is used to represent the maximum number of characters allowed to represent the simple data element.

2) Simple data elements with predefined values (e.g., the simple data element geographic area code[2]): the particularity of this kind is that, values have to be taken from an agreed list of code values. To model the list of predefined codes, we used an OWL class `CodeList` and an object property (`containsCode`) to specify codes belonging to the codes list. Furthermore, an OWL class `Code` and two data properties (`value` and `interpretation`) are used to specify respectively the value that can take and the interpretation of each value.

### C. Modeling of Composite Data Elements

Composite data elements are concatenations of two or more simple data elements. A composite data element is defined by:

- Its EDIFACT identification code.
- Its name and its description.
- Its composition.

Figure 3 shows the OWL serialization of composite data elements. A composite data element is modeled by an OWL class `CompositeDataElement`. We reused data properties which have been introduced previously (i.e., `hasIdentifier`,`hasDescription` and `name`), and we have introduced one object property (`isComposedOf`) to express the relationship between a composite data element and its components (simple data elements). An OWL class `Component` is introduced to specify the component data elements to be included in the composite data element. Furthermore, some properties are also introduced:

1) `containsComponent`: an object property that specifies the simple data element to be included into the composite data element.
2) `hasStatus`: a data property to specify whether the simple data element is mandatory or optional.
3) `hasPosition`: a data property to specify the sequence of components (simple data elements) in the composite data element.
4) `hasRepeatability`: a data property specifying the repeatability of the component (the maximum number of occurrence).
5) `hasPrevious` and `hasnext`: object properties that specifies respectively, the previous and the next component.

### D. Modeling of Segments

A segment is an ordered list of related data components (simple and/or composite) usually associated in a functional way and thus manipulated as such by the partners of the exchange (the sender and the receiver). For instance, consider the segment address [3] which contains, information about the road, postal code, town, country, etc. Each segment is standardized and is reproduced identically in all messages which use it. A segment is defined by:

- Its EDIFACT code (a three capital letters abbreviation of its name).
- Its name and its function.
- Its composition.

Figure 4 shows the OWL serialization of a segment. A segment is modeled by an OWL class `Segment`. We reused data properties which have been introduced previously (i.e., `hasIdentifier` and `name`), and we have introduced a data property (`hasFunction`) to express the function for which it was defined. We also reused concepts (`Component`), object and data properties defined in the previous subsection to indicate the data element (simple and/or composite) to be included in the segment, the sequence of the components, to indicate whether they are mandatory or optional, and to indicate how many times a particular simple or composite element can be repeated.

### E. Modeling of Messages

Messages are the main structure of the EDIFACT exchange standard. They correspond to specific business messages and cover the needs of different sectors of economic activity (order, invoice, payment order, etc.). A message is defined by:

- Its EDIFACT code (a six capital letters abbreviation of its name).
- Its name and its composition.

Figure 5 shows the OWL serialization of a message. A message is modeled by an OWL class `Message`. We reused

[2]http://www.unece.org/fileadmin/DAM/trade/untdid/d16b/tred/tred3279.htm

[3]http://www.unece.org/fileadmin/DAM/trade/untdid/d16b/trsd/trsdadr.htm

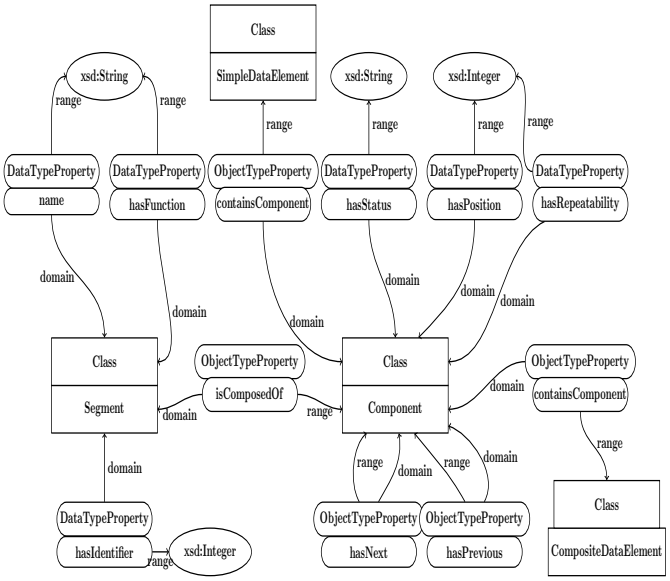


Figure 4. Segment Modeling.

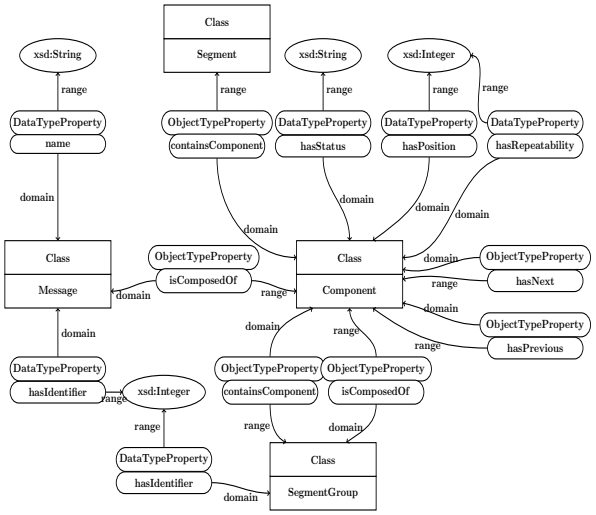


Figure 5. Message Modeling.

data properties which have been introduced previously (i.e., `hasIdentifier` and `name`), and we also introduced an OWL class `SegmentGroup` to represent group of segments. The possibility of grouping segments comes from the need to gather logical sets of information, or to repeat these logics of sets (simple or grouped). Thus, we can constitute groups of hierarchically dependent segments with the possibility that a segment group may contain other groups. We also reused concepts (`Component`), object and data properties defined previously to indicate the segments and/or groups of segments to be included into the message or into the group of segments, the sequence of the components, whether they are mandatory or optional, and to indicate how many times a particular segment or a group can or have to be repeated.

## III. Implementation and Application

Figure 6 shows the high-level architecture of our prototype. Two main steps have oriented our software development process: the conversion of UN/EDIFACT Messages into RDF triplets format and the development of tools to interrogate and visualize the results. The results are represented as a knowledge base (an RDF dataset) which can then be queried using SPARQL queries.

### A. Populating the Ontology

Ontology population is the task of creating individuals (instances) in each class in the OntoEDIFACT ontology, adding data properties between the instances and their literal values, as well as establishing object properties between instances in different classes. The objective of this step is to convert the entire content, syntax and data structures of the EDIFACT standard in order to produce RDF triplets format. To carry out this step, we have developed parsers that apply the strategy defined below and we have used the following frameworks:

- The Jena API [8], which is a free and open source Java framework for building Semantic web applications.
- The Jena TDB triplestore [9], which provides several methods for large scale storage and queries of RDF datasets.
- The JSoup [10] parser, which provides a very convenient Java library for extracting and manipulating data from HTML documents.

All the specifications of the UN/EDIFACT standard are available on the official website [11] of the UNECE. From this website our parsers have extracted the necessary information to populate our ontology. Each year, the UN/EDIFACT standard is reviewed and updated twice. In our prototype, we used the last version (D.16B), which is the second update of the year 2016. Each version of UN/EDIFACT is grouped in four directories:

1) The directory of simple data element[4]: This directory contains 646 HTML pages. Our program start by populating the ontology with simple data elements. For each HTML page specifying a simple data element, an individual 'SE' of class `SimpleDataElement` is created. Afterward, the HTML content is parsed to extract the identifier, the name and the description. These values are associated to the individual 'SE' through data properties (`hasIdentifier`, `name` and `hasDescription`).
If the simple data element has a list of predefined codes, an individual 'CL' of class `CodeList` is also created and linked to the individual 'SE' through the object property `hasCodeList`. For each code that the simple element can take, an individual 'C' of class `Code` is created and linked to the individual 'CL' through the object property `containsCode`. Finally, the value and the interpretation of the corresponding code are extracted and associated to the individual 'C' through the data properties (`value` and `interpretation`).
For simple data elements with free values, an individual 'R' of class `Representation` is created

[4]http://www.unece.org/fileadmin/DAM/trade/untdid/d16b/tred/tredi2.htm

and linked to the individual 'SE' through the object property `hasRepresentation`. The type and the maximum length are then extracted and associated to the individual 'R' through data properties (`type` and `length`).

2) The directory of composite data element[5]: This directory contains 198 HTML pages. For each HTML page specifying a composite data element, an individual 'CE' of class `CompositeDataElement` is created. Then, the HTML content is parsed to extract the identifier, the name and the description. These values are associated to the individual 'CE' through data type properties (`hasIdentifier`, `name` and `hasDescription`). For each component of the composite elements, an individual 'C' of class `Component` is also created and linked to the individual 'CE' through object type property (`isComposedOf`. The repeatability, the position and the status are extracted and associated to the individual 'C' through data properties (`hasStatus`, `hasPosition` and `hasRepeatability`). At last, an object property (`containsComponent`) is used to link the individual 'C' to the simple data element composing the individual 'CE'.

3) The directory of segments[6]: This directory contains 156 HTML pages. For each HTML page specifying a segment, an individual 'S' of class `Segment` is created. Then, the HTML content is parsed to extract the identifier, the name and the function. These values are associated to the individual 'S' through data type properties (`hasIdentifier`, `name` and `hasFunction`). For each component of the segment, an individual 'C' of class `Component` is created and linked to the individual 'S' through object type property (`isComposedOf`). The repeatability, the position and the status are extracted and associated to the individual 'C' through data properties (`hasStatus`, `hasPosition` and `hasRepeatability`). At last, an object property (`containsComponent`) is used to link the individual 'C' to the simple or the composite element composing the individual 'S'.

4) The directory of messages[7]: This directory contains 195 HTML pages. For each HTML page specifying a message, an individual 'M' of class `Message` is created. Then, the HTML content is parsed to extract the name and the identifier. These values are associated to the individual 'M' through data type properties (`hasIdentifier` and `name`). For each component of the message, an individual 'C' of class `Component` is created and linked to the individual 'M' through the object property (`isComposedOf`. The repeatability, the position and the status are extracted and associated to the individual 'C' through data properties (`hasStatus`, `hasPosition` and `hasRepeatability`). At last, an object property (`containsComponent`) is used to link the individual 'C' to the segment or the group of segments composing the individual 'M'. We notice that an individual of class `SegmentGroup` is created for each new group segment encountered, and handled as a component of the message as the same way of segments.

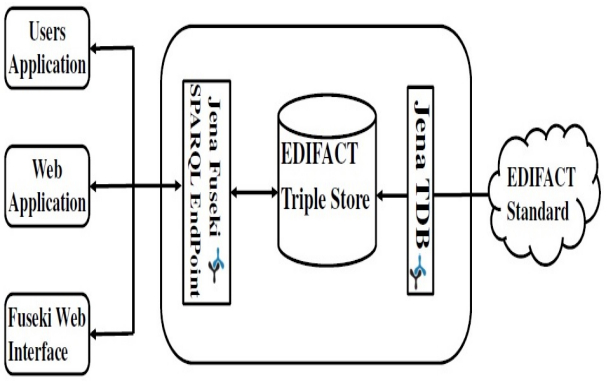


Figure 6. Architecture.

### B. Querying and visualization of the data-set

Once the data set is populated automatically, we have exposed it through three intuitive ways.

The first one is simple Java Web-based [8] application deployed on a WildFly[9] application server (a set of Servlets and JSP pages). It provides a user-friendly interface that helps users to view all the necessary information for designing and creating EDIFACT components by selecting the suitable classes, objects and properties.

The second one is through the Fuseki web user interface[10]. It offers an easy-to-use querying interface that incorporates a lot of functionality. A user can specify SPARQL SELECT queries to directly manipulate the designated components and to view or to download sets of results in several formats (JSON, XML, CSV, etc.).

Finally, for external users applications using traditional framework for querying and analyzing RDF data(Jena [8], Mulgara [11] and Sesame [12] for Java developers or CubicWeb [13] for Python developers).

## IV. Related Work

Despite the exhaustive list of tools proposed by several companies to create XML schema and to convert between EDI standards or formats ([12][13][14]), very few works deal with the issues of ontologization of EDI standard, as illustrated by the research works in [15][16][17]. The authors of [16] propose an ontology for specifying ANSI X12 format. Using this ontology, they encoded the entire version of January 2005 (Data Elements, Composite Data Elements, Segments, and most used Transaction). In order to build this ontology, authors start by specifying the format of X12 components as a classes (Transactions, Segment Groups, etc.), data and

[5]http://www.unece.org/fileadmin/DAM/trade/untdid/d16b/trcd/trcdi2.htm
[6]http://www.unece.org/fileadmin/DAM/trade/untdid/d16b/trsd/trsdi2.htm
[7]http://www.unece.org/fileadmin/DAM/trade/untdid/d16b/trmd/trmdi2.htm
[8]http://realgrain.litislab.fr/EDIFACT-PROJECT/ServletMainEDIFACT
[9]http://wildfly.org/
[10]http://realgrain.litislab.fr:3030/sparql.tpl
[11]http://mulgara.org/
[12]http://rdf4j.org/
[13]https://www.cubicweb.org/

object properties. The process of creation of individuals for each class from HTML files of the X12 specification, is done semi-automatically. In [15], authors propose an ontology codified in OWL to conceptualize the EDIFACT standard. First, authors start by creating classes for each of EDIFACT standard component. Then, they introduce data and object properties to link objects belonging to classes with their values. To populate the ontology, authors developed custom parsers for the (D.97A) version. Comparing with the work of [15], our approach has several advantages. Particularly, that our ontology is generic and can be used to populate the knowledge base with any version of the standard. Parsers for the last version (D.16B) are developed and can be customized for any anterior or future version. Also, in [15] all individual are grouped in one OWL file and this fact can weigh down applications that use this ontology. In our approach, users can choose (through SPARQL requests) only EDIFACT messages that they need in their applications.

## V. Conclusion and Future Works

In this paper, we have presented an ontology OntoEDIFACT (publicly available online [18]) which, has given a semantic representation to the UN/EDIFACT standard. This Ontology has been designed following best practices in ontology engineering. Then, we briefly discuss the functionalities of tools that automatically populate the ontology with instances from the last version of the standard (D.16B). These tools are able to parse HTML web pages containing specifications of UN/EDIFACT components to RDF triples. Finally, a SPARQL endpoint has been implemented and some services have been designed to consume these triples. Of course some improvements can be made to our ontology. For instance, an enrichment phase with external ontologies is underway development. So far, we are aware of the following practical applications that can make use of OntoEDIFACT:

- Indexing UN/EDIFACT documents [19][20] to search efficiently for specific contents inside a large document base.
- Interoperability between the several EDI standards using techniques of ontologies alignment [21]. An investigation work to interoperate the ANSI X12 ontology [16] with the OntoEDIFACT is ongoing.
- Generating XML schema for UN/EDIFACT messages from the OWL representation of each message using straightforward transformation techniques [22].

## Acknowledgment

This research is supported by the ≪CLASSE2≫ project: co-financed by the European Union with the European regional development fund (ERDF) and Normandy Region.

## References

[1] N. C. Hill and D. M. Ferguson, “Electronic data interchange: A definition and perspective,” 1989.

[2] S. Sawabini, “Introduction to edi,” Conference Proceedings EDI 2000: EDI, EC, and You, pp. 1–36.

[3] F. Bergeron and L. Raymond, “The advantages of electronic data interchange,” SIGMIS Database, vol. 23, no. 4, Oct. 1992, pp. 19–31. [Online]. Available: http://doi.acm.org/10.1145/146553.146556

[4] Wikipedia. Edifact-wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/EDIFACT [retrieved: 03, 2017]

[5] A. Gangemi and V. Presutti, “Ontology design patterns,” in Handbook on Ontologies, 2009, pp. 221–243. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-92673-3_10

[6] T. R. Gruber, “Toward principles for the design of ontologies used for knowledge sharing,” Int. J. Hum.-Comput. Stud., vol. 43, no. 5-6, Dec. 1995, pp. 907–928. [Online]. Available: http://dx.doi.org/10.1006/ijhc.1995.1081

[7] M. Horridge and P. Patel-Schneider. OWL 2 web ontology language. manchester syntax (second edition). [Online]. Available: http://www.w3.org/TR/owl2-manchester-syntax/ [retrieved: Dec., 2012]

[8] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, “Jena: Implementing the semantic web recommendations,” in Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers &Amp; Posters, ser. WWW Alt. ’04. New York, NY, USA: ACM, 2004, pp. 74–83. [Online]. Available: http://doi.acm.org/10.1145/1013367.1013381

[9] JenaTDB. Apache jena - tdb. [Online]. Available: https://jena.apache.org/documentation/tdb/ [retrieved: 03, 2017]

[10] Jsoup. jsoup: a java html parser library. [Online]. Available: https://jsoup.org/ [retrieved: 03, 2017]

[11] EDIFACT. Edifact directories. [Online]. Available: https://www.unece.org/tradewelcome/un-centre-for-trade-facilitation-and-e-business-uncefact/outputs/standards/unedifact/directories/2011-present.html [retrieved: 03, 2017]

[12] Altova. Xml global and altova reduce data integration costs with the launch of the enterprise-ready xml integration workbench. [Online]. Available: https://www.altova.com/ [retrieved: 03, 2017]

[13] Smooks. The smooks framework. [Online]. Available: http://www.smooks.org/ [retrieved: 03, 2017]

[14] S. Studio. The stylus studio edi to xml. [Online]. Available: http://www.stylusstudio.com/edi/ [retrieved: 03, 2017]

[15] R. Engel, C. Pichler, M. Zapletal, W. Krathu, and H. Werthner, “From encoded edifact messages to business concepts using semantic annotations,” in Proceedings of the 2012 IEEE 14th International Conference on Commerce and Enterprise Computing, ser. CEC ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 17–25. [Online]. Available: http://dx.doi.org/10.1109/CEC.2012.13

[16] D. Foxvog and C. Bussler, “Ontologizing edi: First steps and initial experience,” in Proceedings of the International Workshop on Data Engineering Issues in E-Commerce, ser. DEEC ’05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 49–58. [Online]. Available: http://dx.doi.org/10.1109/DEEC.2005.13

[17] D. B. Foxvog and Christoph, “Ontologizing edi semantics,” in Proceedings of the 2006 International Conference on Advances in Conceptual Modeling: Theory and Practice, ser. CoMoGIS’06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 301–311. [Online]. Available: http://dx.doi.org/10.1007/11908883_36

[18] T. E. Ontology. The edifact ontology. [Online]. Available: http://realgrain.litislab.fr/OntoEDIFACT.owl [retrieved: 03, 2017]

[19] E. Desmontils and C. Jacquin, “Indexing a web site with a terminology oriented ontology,” in Proceedings of the First International Conference on Semantic Web Working, ser. SWWS’01. Aachen, Germany, Germany: CEUR-WS.org, 2001, pp. 549–565. [Online]. Available: http://dl.acm.org/citation.cfm?id=2956602.2956638

[20] F. Fürst and F. Trichet, “Integrating domain ontologies into knowledge-based systems,” in Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference, Clearwater Beach, Florida, USA, 2005, pp. 826–827. [Online]. Available: http://www.aaai.org/Library/FLAIRS/2005/flairs05-142.php

[21] S. Pavel and J. Euzenat, “Ontology matching: State of the art and future challenges,” IEEE Trans. on Knowl. and Data Eng., vol. 25, no. 1, Jan. 2013, pp. 158–176. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2011.253

[22] I. Bedini, C. Matheus, P. F. Patel-Schneider, A. Boran, and B. Nguyen, “Transforming XML schema to OWL using patterns,” in ICSC 2011 - 5th IEEE International Conference on Semantic Computing, Palo Alto, United States, 2011, pp. 1–8. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00624055

 ISBN: 978-1-61208-558-6 