Semantic EDI

Negar Batenipour, Fardis Amouzadeh Araei, Kristina Enes

University of Bonn

Abstract. The traditional method of communication between two business parties is paper-based, but it has many disadvantages regarding time and human interventions. To overcome these issues, we use Electronic Data Interchange (EDI) systems to transmit business messages in a compact form between business parties. But there are still weak points, such as high processing time and high human effort; therefore, we propose Semantic EDI, as a new method, which represents EDI messages in a machine-interpretable format (RDF). Furthermore, we developed a Smart Client application to demonstrate the benefits of this format.

1 Introduction

Electronic Data Interchange (EDI) systems are used to transmit business messages in a compact form between enterprises electronically, against the traditional method of communication on paper, such as purchase orders and invoices. EDI has different standards, including X12, EDIFACT, ODETTE, etc [1].
EDI, as an electronic interchange, in comparison to paper-based interchange, has less human interventions, less human errors, less manual entries, which causes less labor hours. It results in a better B2B (business to business) relationship between business partners, which causes mutual cost saving. Besides those benefits, EDI has its own weak points, such as high processing time and high human effort. In Fig.1, you can see the processing of an order without EDI and with EDI.
For overcoming the weak points of EDI, we propose translating EDI messages into semantic forms in which the statements encoded in the compact forms are explicitly expressed [2].
For this purpose, we have written the code in Python programming language to design the translator, which converts EDI messages into the Turtle serialization. In our project, we have created a complete sample of an EDI ordering message and adjusted our code for two versions of EDI ordering messages (2003 and 2016), whereas the similarities between all versions are great.
Furthermore, we have designed a Smart Client to demonstrate the use of our translator, which performs SPARQL queries on the generated RDF EDI Message and a Turtle file containing information about the purchased product in the original EDI message and its production process.

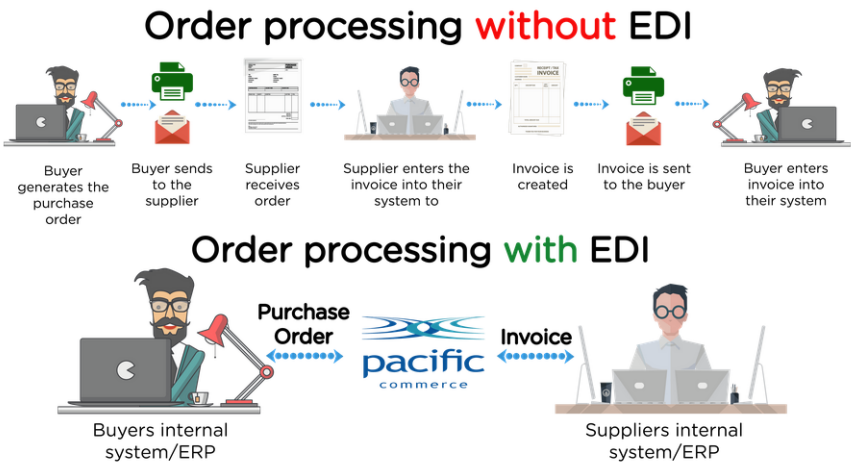

*Fig.*1

2 Background

There are different standards available for EDI, one of those is EDIFACT (Electronic Data Interchange for Administration, Commerce and Transport). EDIFACT is the international standard for EDI and provides rules how an EDI messages has to be structured.
Fig.2 shows an example of a purchase order. The corresponding EDI Order messages is shown in Fig.3. There are several types of EDI messages, each defined by a six character name. Examples are ORDERS (Purchase Order Message) or INVOIC (Invoice Message). We will focus solely on purchase order messages.
An EDI message consists of several segments, divided by '. In Fig.2 each line is a segment. Each segment starts with a three character code. Examples are BGM (Beginning of Message) or NAD (Name and Address). Each EDI message starts with a UNH (Message Header) segment and ends with a UNT (Message Trailer) segment.
A segment is further divided into data elements, either single or composite. A single data element contains one piece of information, a composite data element can contain more pieces of information. Data elements are divided by + and the pieces of informations in a composite data element are divided by : [4].
Using this type of interchange has several disadvantages:

- Longer processing time is needed in sequenced-format information.
- Because of its string-format, errors increase.
- It has slow access time for retrieving documents.

Using Semantic EDI instead, we have an explicit representation of all information in a messages, which overcomes mentioned disadvantages. This format is

based on ontologies and turtle triples, which is easy to read and doesn't have EDI's defects.

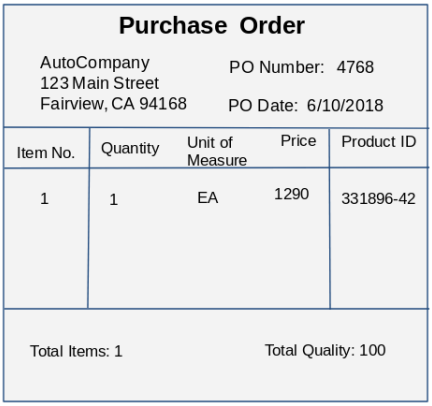

Fig.2

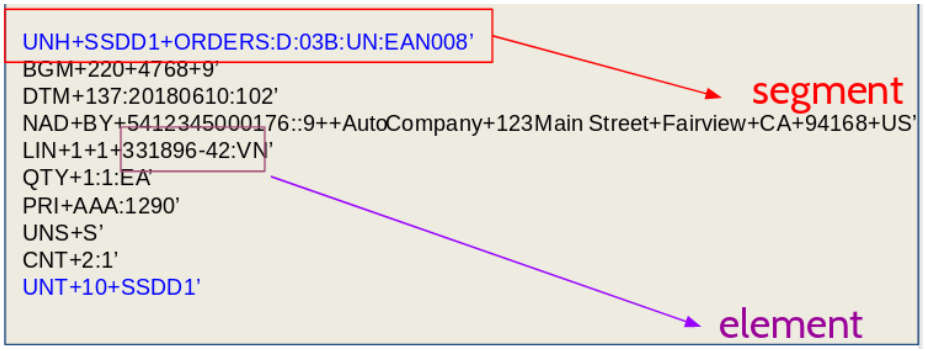

Fig.3

3 Related Work

There are not many works regarding semantic EDI. In [2] the authors, too, translated EDI messages into a semantic form. But instead of using the EDIFACT standard, they based their translator on messages of the ANSI X12 EDI standard.
In a Bachelor Thesis [5] EDIFACT INVOIC (invoice messages) were translated. In this case the tool BOTS [6] was used to represent the EDI message in a tree. Using a mapping script, this tree is searched for relevant information and written in a Semantic EDI document. We have instead directly written a translator to transform EDI messages into Semantic EDI messages, assuming that all information in the original message are relevant.

4 Proposed Approach

In our approach, we translate the EDI ordering messages into Semantic EDI messages in turtle serialization format. We focus on EDIFACT messages of the type ORDERS. For translating those EDI messages we consider all possible segments, elements and components that can appear in a purchase order message.

4.1 Problem Statement

The Electronic Data Interchange (EDI) sequenced-based format, has several weak points; for example, when an EDI ordering message is sent by the sender party, the processing time of this message on the receivers end is long, the access time for retrieving the ordering message is slow and errors increase because of its sequenced-based format.
Furthermore both business partners have to decide on the specific syntax of the message, they have to agree on how and where specific information should appear in an EDI message. Depending on the number of business partners this takes a lot of work and time. This can also result in misinterpretation of the EDI message [5].
If there is a semantic form of the EDI message, misinterpretations are less likely to occur, and the correct processing of those messages will be faster.

4.2 Proposed Solution

We propose Semantic EDI, Turtle serialization triples of EDI order messages. Because of its machine-interpretable format, we can decrease the processing time, improve the access time for retrieving documents, and reduce the errors.

4.3 Implementation

We have written our code in Python to build a translator, which converts EDI ordering message into ontologies in turtle serialization format. We have considered the two versions of EDI ordering message, EDI2003 (D.03B) and one of the more recent versions, EDI2016 (D.16B), but almost all of the versions are quite similar [3].
In Fig.4, a part of our code has been represented. It show the translation of an IMD segment (Item Description).

In the following, we have compared these two versions in the number of segments, elements and components:
- D.03B: 49 segments, 194 elements, 364 components
- D.16B: 52 segments, 214 elements, 412 components

In both versions, some segments have a different number of elements and components, such as in the following segments: BGM, TAX, TDT, EQD, and DGS. Some segments have different names for elements and components, such

as LIN, RCS, PCI, RFF, NAD, FII, CTA, and COM. In D.16B version, new segments, such as EFI, CED, and STS have been added. In general most version are quite similar. So our translator is capable of converting the most basic EDI order messages, no matter the version. But errors might still occur if our translator takes more complex EDI order messages of different versions as input.
To test our translator we have created EDI ordering messages as an input both EDI versions. These messages contain all possible segments that can appear in an EDI order message of the respective version.

```
789         #IMD
790         if IMDElements >= 2:
791             g.add((Order1, n.DescriptionFormatCode, Literal(IMD[1][0])))
792         if IMDElements >= 3:
793             if IMDComponents[2] >= 1:
794                 g.add((Order1, n.ItemCharacteristicCode, Literal(IMD[2][0])))
795                 if IMDComponents[2] >= 2:
796                     g.add((Order1, n.CodeListIdentificationCode, Literal(IMD[2][1])))
797                     if IMDComponents[2] >= 3:
798                         g.add((Order1, n.CodeListResponsibleAgencyCode, Literal(IMD[2][2])))
799         if IMDElements >= 4:
800             if IMDComponents[3] >= 1:
801                 g.add((Order1, n.ItemDescriptionCode, Literal(IMD[3][0])))
802                 if IMDComponents[3] >= 2:
803                     g.add((Order1, n.CodeListIdentificationCode, Literal(IMD[3][1])))
804                     if IMDComponents[3] >= 3:
805                         g.add((Order1, n.CodeListResponsibleAgencyCode, Literal(IMD[3][2])))
806                         if IMDComponents[3] >= 4:
807                             g.add((Order1, n.ItemDescription, Literal(IMD[3][3])))
808                             if IMDComponents[3] >= 5:
809                                 g.add((Order1, n.ItemDescription, Literal(IMD[3][4])))
810                                 if IMDComponents[3] >= 6:
811                                     g.add((Order1, n.LanguageNameCode, Literal(IMD[3][5])))
812         if IMDElements >= 5:
813             g.add((Order1, n.SufaceOrLayerCode, Literal(IMD[4][0])))
```

*Fig.*4

5 Use Cases

A company produces several kinds of products. After receiving a semantic EDI ordering message from a customer, the company can check if enough of the ordered products are available or not. If not enough of those products are in stock they have to be produced in a short amount of time. Given information about the production process the time needed to produce the remaining products can be calculated.

6 Evaluation

We have designed a Smart Client to demonstrate the use of our translator. The Smart Client performs queries on turtle files using SPARQL.

In our Smart Client, we consider three different ordering messages. Each of them has different Item Identifier, Quantity, and Price; therefore, our given inputs are three output.ttl files (outputs of translator for three different EDI ordering messages) and data.ttl. Data.ttl contains information about the company and its products like the production process and the corresponding production line of each product. It also contains information about the machines needed to produce a product and when those machines are free and how long it takes to produce the product once. In Fig. 5, a part of the input of our code, data.ttl is represented. In Fig. 6 the output of our Smart Client is shown. The Smart Client makes several inquiries:

- What is the Item Identifier of the ordered products?
- What will the company be paid for those products?
- What is the amount of products ordered?
- How many of those products are already available?

If there are not enough products available, the number of products, that still have to be produced is computed. Then the Smart Client calculates the time needed to produce those product, independent of the availability of the machines needed in the production process. In the end, the time until the machines are ready to use is added to the time needed for production, resulting in the time needed until the new products are ready for transport.
Given the RDF format of the EDI message and our data, making those calculation was easy, because the Smart Client is able to interpret the relevant information immediately.

You can find our codes in GitHub:
https://github.com/NegarBatenipour/Semantic-EDI/

```
ns1:Product1 ns1:ItemIdentifier "331896-42";
             ns1:AvailableInStock "1000"^^xsd:positiveInteger;
             ns1:ProductionProcess ns1:Process1 .

ns1:Product2 ns1:ItemIdentifier "331896-43";
             ns1:AvailableInStock "20"^^xsd:positiveInteger;
             ns1:ProductionProcess ns1:Process2 .

ns1:Product3 ns1:ItemIdentifier "331896-44";
             ns1:AvailableInStock "0"^^xsd:positiveInteger;
             ns1:ProductionProcess ns1:Process3 .
```

*Fig.*5

Product	ItemIdentifier	Price	Quantity	AvailableInStock
http://example.org/Product1	331896-42	1290	5	1000
http://example.org/Product2	331896-43	1000	3	20
http://example.org/Product3	331896-44	900	6	0

ReadyForTransport	ProductsToProduce	ProductionTime	NewProductsReadyIn
True	0		
True	0		
False	6	18	28

*Fig.*6

7 Conclusions and Future Work

To transform EDI order messages into Semantic EDI order messages, we developed a translator in Python, one for the EDIFACT version D.03B and one for the D.16B version. For this purpose, we had to consider each possible segment, data element and component, some of them are mandatory and some of them are conditional. Given our first translator for version D.03B, we discovered a few things, while writing the translator for a second version:
The majority of segments, elements and components between two versions are the same. This means that one translator should be able to correctly transform the majority of EDI order messages, independant of the version, but there are a few differences between all of them. They can differ in the numbers of segments, elements and components, or their names; therefore, the translator fails, if given an EDI order message of a different version containing those differences.
In the future, our code could be extended to correctly deal with all possible EDI messages, independant of the version.
In addition our translator can only work with messages of type ORDERS. Currently there are 195 different types of messages. In the future translators could be build for each of those message types.

8 References

[1] Wikipedia, $https://en.wikipedia.org/wiki/Electronic_data_interchange$
[2] D. Foxvog and C. Bussler, Ontologizing EDI Semantics , International Conference on Conceptual Modeling, 2006.
[3] Truugo, $https://www.truugo.com/edifact/d16b/orders/$
[4] Edifactory, $https://www.edifactory.de/edifact/about-edifact$
[5] R. Isaev, "Towards: Improving EDI with Semantic Web", Bachelor Thesis, 2017.
[6] Bots, $https://bots.readthedocs.io/en/latest/$