Automation of Electronic Invoice Validation using Knowledge Graph Technologies

Johannes Mäkelburg[1][0009-0001-3821-7817], Christian John[2], and Maribel Acosta[1][0000-0002-1209-2868]

[1] TUM School of Computation, Information and Technology, Technical University of Munich, Heilbronn, Germany `firstname.lastname@tum.de`
[2] Einkaufsbüro Deutscher Eisenhändler GmbH, Wuppertal, Germany `christian.john@ede.de`

Abstract. Invoicing is a crucial part of any business's financial and administrative activities. Nowadays, invoicing is handled in the form of Electronic Data Interchange (EDI), where invoices are managed in a standardized electronic or digital format rather than on paper. In this context, EDI increases the efficiency of creating, distributing, and processing invoices. The most used standard for representing electronic invoices is EDIFACT. Yet, the validation of EDIFACT invoices is not standardized. In this work, we tackle the problem of automatically validating electronic invoices in the EDIFACT format by leveraging KG technologies. The core of our proposed solution consists of representing EDIFACT invoices as RDF knowledge graphs (KGs). We developed an OWL ontology to model EDIFACT terms with semantic descriptions. The invoice KG can be validated using SHACL constraints acquired from domain experts. We evaluated our ontology and invoice validation process. The results show that our proposed solution is complete, correct, and efficient, and significantly undercuts the efforts of current human evaluation.

Keywords: Electronic Invoice, Ontology, EDIFACT, RDF, RML, SHACL

1 Introduction

In the current business landscape, efficient financial and administrative practices are paramount for organizations. Central to these operations is the invoicing process, a critical component that demands accuracy, timeliness, and standardization. Organizations increasingly rely on Electronic Data Interchange (EDI) to streamline invoicing, transitioning from traditional paper-based methods to electronic formats. Within EDI, EDIFACT is a widely adopted standard for representing electronic invoices, offering a uniform approach to their creation, distribution, and processing.

While EDIFACT has significantly enhanced the efficiency of invoicing procedures, a notable gap exists in the validation of electronic invoices. This gap affects business processes that rely on correct invoicing data in a timely manner. This is the case of the group purchasing organization Einkaufsbüro Deutscher

Eisenhändler (E/D/E). As E/D/E is represented in 30 European countries, where many different processes and regulations exist for business documents, EDIFACT is used as the standard to handle invoices. Yet, as the validation of EDIFACT invoices is not standardized – i.e., there is no language for representing constraints over the invoices – in many cases, this process is carried out manually, which is time-consuming and prone to errors.

This work proposes a solution that leverages KG technologies to validate electronic invoices in the EDIFACT standard. The core of our approach is to model electronic invoices as RDF graphs. Shifting to the semantic web technology stack allows for applying existing open standards and solutions for managing machine-readable data. To achieve this, first, we present the EDIFACT Ontology. The ontology captures the terms to model the content of EDIFACT messages using RDF. Second, we propose the tool EDIFACT-VAL to validate the content of the original EDIFACT messages. The tool processes the invoices in the EDIFACT format and translates them into XML. From the XML files, EDIFACT-VAL creates the RDF graphs using the EDIFACT Ontology and the RDF Mapping Language (RML) [4]. EDIFACT-VAL validates the invoice RDF graph using constraints defined in the Shapes Constraint Language (SHACL) [13]. The constraints are created with input from domain experts based on the EDIFACT guidelines.

We evaluate the soundness of our proposed solution with an experimental evaluation using real-world EDIFACT invoices. The results show that EDIFACT-VAL produces complete RDF graphs in the order of seconds. The validation with SHACL shows that many real-world invoices do not fully comply with the EDIFACT standard. Lastly, we conducted an in-use evaluation with domain experts, who compared EDIFACT-VAL with the current manual process. The experts found the tool's performance remarkable, and are working on integrating it into their invoice workflows. This shows the potential impact of our proposed solution.

In summary, our contributions are:

- An OWL ontology to represent terms from the EDIFACT standard to model electronic invoices as RDF KGs.
- A tool dubbed EDIFACT-VAL to automatically validate invoices using SHACL.
- An experimental evaluation that shows the soundness of our proposed solutions.
- An in-use evaluation where domain experts at the E/D/E assessed the performance and applicability of EDIFACT-VAL to their business processes.

2 Preliminaries

First, we introduce the EDIFACT invoice concept, which forms the basis for the knowledge graphs. Then, we introduce the purchase-to-pay ontology, which builds the basis for an EDIFACT ontology.

EDIFACT Invoice EDIFACT is the most commonly used and most comprehensive international standard for electronic data interchange. EDIFACT is used

Listing 1.1. Excerpt of an EDIFACT invoice

```
1 UNH+1+INVOIC:D:96A:UN:EAN008’
2 BGM+380+4031541+43’
3 DTM+137:20220908:102’
4 NAD+IV+4317784000000::9++Einkaufsbuero DeutscherEisenhaendler:GmbH+EDE PLatz 1+Wuppertal
      ++42389+DE’
5 LIN+1++4016671029277:EN::9’
6 PRI+AAA:16.78:::1:PCE’
7 MOA+79:100.68’
8 MOA+124:19.13’
9 UNT+37+1’
```

across almost all business sectors; the individual sectors are delimited in EDIFACT by so-called subsets. The maintenance of the standard lies under the responsibility of the United Nations and the Economic Commission for Europe.

Documents transmitted in EDIFACT are all types of messages of the business processes area. The structure of the messages is based on segments; these, in turn, consist of data elements and data element groups. These three components together are referred to as the EDIFACT elements.

Listing 1.1 shows an excerpt of a real-world EDIFACT message. The segments are split into three sections: header-, detail- and summary section. In all three sections, some segments are required, meaning all three sections are always represented in an EDIFACT message. However, there are some segments within the sections that are optional. For example, the header section contains eight mandatory segments and six conditional segments.

In the header, general information about the invoice is displayed, e.g., the invoice number (Line 2), the document date (Line 3), and information about the involved organizations (Line 4). Information about the sold items, including the net price (Line 6) or the article number (Line 5), is allocated in the detail section. The summary section contains the total amounts of the invoice, e.g., the total item amount (Line 7) or the total tax amount (Line 8). Above the header and below the summary section are segments allocated for the EDIFACT structure, e.g., the version and type of message (Line 1) or the end character (Line 9).

P2P-O: Purchase to Pay Ontology We use the Purchase-to-Pay Ontology (P2P-O) [21] as a foundation for modeling concepts from the EDIFACT standard. P2P-O is an OWL ontology that models semantic representation of invoices based on the *core invoice model* of the European Standard EN 16931-1:2017 [6].

P2O-O is divided into seven modules: *item, price, documentline, organization, document, invoice, process.* The *item* module allows for describing products listed on the invoices. Not only sold items are mentioned with the term, but also working hours. The *price* module makes it possible to describe the prices of the items and the amounts of money in an invoice. The possibility of making statements about the positions of prices and items on documents is enabled by the model *documentline.* The participating organizations are described by the module *organization.* In the *document* module ontology resources which are essential for the purchase-to-pay process are provided. The document-type invoice

Table 1. Competency Questions for the EDIFACT Ontology

Name	Competency Question
CQ 0	What invoices are all listed in an EDIFACT message?
CQ 1	Which organizations are involved in the invoice?
CQ 2.1	What role does organization S play in the invoice?
CQ 2.2	Which organization is the buyer in the invoice?
CQ 3.1	What information is displayed about the involved organizations ?
CQ 3.2	What is the address of the buyer?
CQ 4	What items are sold in the invoice?
CQ 5.1	What information is displayed about the items sold?
CQ 5.2	What is the net price of the items sold in the invoice?
CQ 6.1	What are the invoice details of the invoice?
CQ 6.2	What is the invoice amount of the invoice?
CQ 6.3	What is the invoice number?
CQ 7	What information must be provided so that the file format is valid?
CQ 8	To which business process can the invoice be assigned?

is described more specifically in the module *invoice*. Therefore the two modules *document* and *documentline* are used. In the *invoice* module, all the mandatory constraints from EN 16931 are implemented. Lastly, the *process* module contains classes for a specific description of the kind of purchase-to-pay process.

3 Our Approach

Given an electronic invoice in the EDIFACT format, the problem tackled in this paper is to validate the correctness of the invoice by representing the invoice as an RDF knowledge graph (KG). Our proposed solution comprises two parts: (1) a proposed ontology to represent EDIFACT concepts (§ 3.1), based on the concepts presented in Sect. 2, and (2) the EDIFACT-VAL tool to perform the validation of invoices using KG technologies (§ 3.2).

3.1 EDIFACT Ontology

Competency Questions We use the NeOn method [25] as a systematic approach to structure and develop the EDIFACT ontology according to an established principle. The ontology development process involves the formulation of the requirements, framework, and competency questions of the ontology. In this work, these steps were carried out in collaboration with Electronic Data Interchange experts from E/D/E. Listing 1 shows the competency questions have been drawn up based on the EDIFACT guideline. The competency questions (CQ) are grouped according to different aspects of the invoices. CQs in groups 1 and 2 (i.e., CQ 1, 2.1, and 2.2) are concerned with organizations and their role in the invoice. CQ 3.1 and 3.2 address specific information about the involved organizations. CQ 4, 5.1, and 5.2 request information about the items listed in the invoice. CQ6 handles the identifier or number of the invoice. Lastly, CQ 7 and 8 address aspects of the invoice metadata, i.e., the EDIFACT structure elements and related business processes.

Table 2. Overview of linked ontologies and vocabularies in the EDIFACT Ontology

Prefix Name	**Prefix**	**Frequency of use**
agentRole	`https://archive.org/services/purl/domain/modular_ontology_design_library/agentrole#`	5
dc	`http://purl.org/dc/elements/1.1#`	2
frapo	`http://purl.org/cerif/frapo/`	3
schema	`http://schema.org/`	3
org	`http://www.w3.org/ns/org#`	1
p2p-o-doc-line	`https://purl.org/p2p-o/documentline#`	2
p2p-o-doc	`https://purl.org/p2p-o/document#`	1
p2p-o-inv	`https://purl.org/p2p-o/invoice#`	3
p2p-o-item	`https://purl.org/p2p-o/item#`	2
p2p-o-org	`https://purl.org/p2p-o/organization#`	4
vcard	`http://www.w3.org/2006/vcard/ns#`	2

Reuse of Ontology Design Patterns and Existing Vocabularies and Ontologies Following ontology design best practices [25, 17], we reuse existing resources. An overview of the reused resources and the number of reuses can be found in Table 2. We apply the agent role pattern from the Modular Ontology Design Library (MODL) [22] for modelling the participation of organizations in invoices. In particular, the same organization can have many roles (seller, supplier, etc.) in the same or different invoices, to which different property values can be associated depending on its role. The ontologies listed in Table 2 also provide adequate solutions for our purpose. Most of them are reused in the class *AgentRole*, for instance, the country code from the Funding, Research Administration, and Projects Ontology [23] or the address of the vCard Ontology [16]. Also, four of the seven different main classes we defined in our EDIFACT Ontology are linked to concepts of these vocabularies or ontology. For example, the class *FormalOrganization* is linked to the Core Organization Ontology (*org*), the *E-Invoice* to the P2P-O module *document*, and *Item* to the P2P-O module *item*.

Ontology Description Based on the gaps identified in existing vocabularies and ontologies discussed in the previous section, we propose the EDIFACT Ontology tailored to represent the concepts and fields of the EDIFACT standard. The proposed OWL ontology comprises 28 classes, 10 OWL object properties, 233 OWL data properties, and 6 annotation properties. The ontology was developed using WebProtégé [11]. Figure 1 illustrates the main concept of the EDIFACT ontologies by showing the connections between the different classes. The EDIFACT Ontology can be found under `https://purl.org/edifact/ontology`. Additionally, it has been integrated into the LOV catalogue, available at `https://lov.linkeddata.es/dataset/lov/vocabs/edifact-o`. The classes, their properties, and how they address the competency questions from Listing 1 are explained in more detail in the following.

E-Invoice This is the main class of the EDIFACT ontology, and its individuals or entities represent electronic invoices. This class is connected to other classes through object properties, including, *EDIFACT-Structure* via the property *followsStandard*, *Item* via the property *hasItem*, *InvoiceDetails* via the property *hasInvoiceDetails*, and *AgentRoles* via the *isProvidedBy* property to capture the

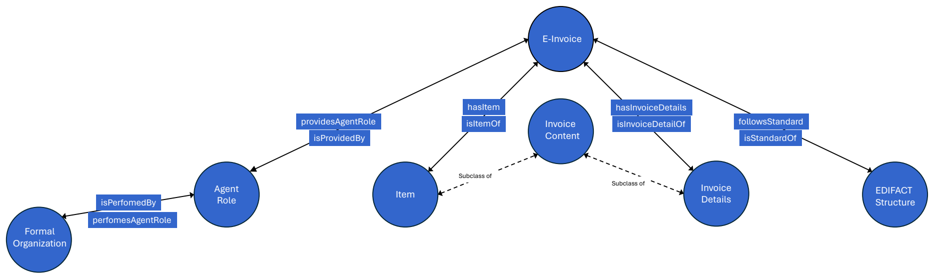

Fig. 1. Main concepts of the EDIFACT ontololgy. Source: WebVOWL [15]

role of organizations in the invoice. The only data property used to describe this class is *belongsToProcess* which displays the business process of the invoice. This class allows for handling the competency questions CQ 0 and CQ 8.

EDIFACT-Structure This class contains all the information that ensures that the invoice file meets the requirements of the EDIFACT file format. This information appears at the beginning and end of a message and, therefore, has no correspondence with the content of the individual invoices. For this reason, this class and the *InvoiceContent* class are disjoint, specified with the *owl:disjointWith* predicate. Nevertheless, as this information belongs to an invoice, this class is connected to the class *E-Invoice* via the object property *followsStandard*. Among the datatype properties of the entities of this class, we have *creationDate*, *dataExchangeCounter*, *messageReferenceNumber*, and *senderIndicator*. This class is associated with the competency question CQ 7.

Invoice Details This class contains all the information that can only occur in the header- and summary sections of the invoices. Exemplary datatype properties of this class are the document date (*hasDocumentDate*), the document number (*hasDocumentNumber*), the delivery condition (*deliveryCondition*) and the invoice amount (*hasInvoiceAmount*). The connection of the *InvoiceDetails* class to the *E-Invoice* class is done via the object property *hasInvoiceDetails*. The class *Invoice Details* allows handling the competency questions CQ 6.1, 6.2, and 6.3.

Item This class allows for representing an item listed in the invoice, which is found in the detail section of the invoice. Examples of the datatype properties for an item are the name of the item (*p2p-o-item:Item*), the net price of the item (*hasNetPriceOfItem*), the net weight of an item (*hasNetWeight*), the number assigned to a manufacturer's product according to the International Article Numbering Association (*internationalArticleNumber*), etc. It is to mention that the ratio between the classes *E-Invoice* and *Item* is 1:N. Therefore, here we were confronted with the design decision of modeling the relationship between these two classes as a multi-valued property or as an RDF collection to represent a close list. We decided on the former option by defining the object property *hasItem* (and its inverse *isItemOf*) between the *Item* and the *E-Invoice* classes, as this facilitates the validation and querying of the invoices. The *Item* concept allows for handling the competency questions CQ 4, 5.1, and 5.2.

Listing 1.2. Representation of organizations using *AgentRole* and *FormalOrganization*

```
@prefix invoice: <http://www.ede.com/edifact/invoice#>.
@prefix edifact-o: <https://purl.org/edifact/ontology#> .
@prefix agentRole: <https://archive.org/services/purl/domain/modular_ontology_design_library/
     agentrole#> .
@prefix frapo: <http://purl.org/cerif/frapo/> .
@prefix p2p-o-org: <https://purl.org/p2p-o/organization#> .
@prefix vcard: <http://www.w3.org/2006/vcard/ns#> .

invoice:r999995 a edifact-o:DeliveryPartyRole;
  frapo:hasCountryCode "DE";
  edifact-o:hasCity "Wuppertal";
  edifact-o:hasCountry "Deutschland";
  vcard:hasStreetAddress "In der Fleute 153";
  vcard:postalCode "42389";
  agentRole:isProvidedBy invoice:i999999;
  p2p-o-org:formalName "E/D/E-GmbH Anlieferstelle L205" .

invoice:r999996 a edifact-o:InvoiceeRole;
  frapo:hasCountryCode "DE";
  edifact-o:hasCity "Wuppertal";
  edifact-o:hasCountry "Deutschland";
  vcard:hasStreetAddress "EDE Platz 1";
  vcard:postalCode "42389";
  agentRole:isProvidedBy invoice:i999999;
  p2p-o-org:formalName "Einkaufsuero Deutscher Eisenhaendler".

invoice:4317784000000 a edifact-o:FormalOrganization;
  agentRole:performsAgentRole invoice:r999995, invoice:r999996 ;
  p2p-o-org:globalLocationNumber 4317784000000 .
```

InvoiceContent This class allows for modelling the information that can be found in all three sections of the invoices. As a result, the information of the *Invoice-Content* class is defined as the union of the classes *Item* and *InvoiceDetails*, i.e., *InvoiceContent owl:unionOf (Item InvoiceDetails)*. Creating the *InvoiceContent* class makes it possible to define properties that apply to all parts of the invoice by using this class as the domain or range of those properties. This ensures the consistency of the ontology.

AgentRole According to the guidelines, an organization can, or sometimes must, have many different roles. For example, in the warehousing business, one organization needs to have three roles: *Buyer Role*, *Invoicee Role*, and *Delivery Party Role*. We model these cases using the Role ontology pattern as defined by Shimizu et. al [22] and Grüninger & Fox [8]. In our ontology, the purpose of the class *AgentRole* is to represent the manifold roles an organization can have in an invoice. As a solution, several subclasses have been created, each representing one of these roles. The connection to the *E-Invoice* class exists through the object property *isProvidedBy* with the *E-Invoice* class in the range. The *AgentRole* class allows for addressing the competency questions C 2.1, 2.2, 3.1, and 3.2.

FormalOrganization This class represents the organizations involved in the messages, which addresses the competency question CQ 1. The properties assigned to the *FormalOrganization* class are solely responsible for allocating the role that an organization plays in an invoice through *performsAgentRole*, and for provid-

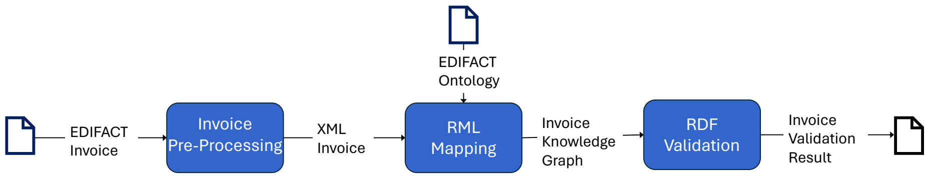

Fig. 2. Overview of EDIFACT-VAL

ing a globally unique identifier for organizations, i.e., *globalLocationNumber*. In the EDIFACT ontology, the only connection of the *FormalOrganization* class with another class is with the *AgentRole* class via the object property *performsAgentRole* with the *AgentRole* class in the range.

We have opted against the intuitive option of assigning the information regarding the organization to the class of *FormalOrganization*. Instead, we assign them to the class *AgentRole*, more precisely to one of its subclasses. This is because some of the displayed information can change for the same organization depending on its role in an invoice. We illustrate this in the following example.

Listing 1.2 showcases the RDF triples of the different agent roles related to warehouse business invoices. The address of the organization changes depending on its role: the address of the warehouse *In der Fleute 153* (Line 12) is used when the organization has the DeliveryParty Role, while the main location *EDE Platz 1* (Line 21) is used for the Invoicee Role. However, all roles belong to the same organization, as evidenced by both agent roles being performed by the same *FormalOrganization* (Line 27 and 28).

3.2 The EDIFACT-VAL Tool

Now that we have modelled the terms of the EDIFACT standard in an OWL ontology (see Sect. 3.1), the EDIFACT-VAL tool processes the electronic invoices in the EDIFACT format to validate their content using KG technologies. Concretely, the tool carries out the following steps (cf. Figure 2). **(1) Invoice Preprocessing:** Translates the invoice files into XML files. **(2) RML Mapping:** Creates RDF knowledge graphs from the invoice XML files using generated RML mappings. **(3) RDF Validation:** Validates the invoice RDF graphs using constraints based on EDIFACT guidelines and reports from domain experts, which are formulated using the Shapes Constraints Language (SHACL) [13].

Invoice Pre-Processing This step aims to prepare the EDIFACT message in a way that meets the requirements for the creation of an RDF graph. Despite that EDIFACT is an open standard, the file format is not yet widely compatible with existing tools, especially the ones related to knowledge graphs. Hence, a transformation into a compatible format is required. Our choice of format is XML, justified by the ease of modification and the numerous processing possibilities. Then, EDIFACT-VAL extends the resulting XML files, to incorporate additional XML elements and attributes that capture the qualifiers in EDI files.

EDI files use qualifiers to secure file compactness, i.e., several values are packed into a single field. An example of this is shown in Line 4 of Listing 1.1. There the NAD-segment with the qualifier *IV* is used to display information about the Invoicee of the invoice. To handle such qualifiers, the pre-processing separates this field into several XML elements, creating a one-to-many correspondence between EDIFACT elements and XML elements.

RDF Graph Creation for Invoices EDIFACT invoices can be transformed into RDF representations using our proposed EDIFACT Ontology described in Sect. 3.1. To generate RDF knowledge graphs based on a (semi-structured) data source, we define mapping rules. For this, we use the RDF Mapping Language (RML) [4], which provides a way to transform heterogeneous data into the RDF data model. Yet, manually creating RML mapping is a complex task [12], especially when a large number of terms are involved. To ease the definition of the mapping rules, languages like YARRRML [26] and ShExML [7] provide a human-friendly serialization of RML. In this work, we use YARRRML [26] as it allows users to specify simpler mapping rules compared to RML;[3] these mappings are then automatically transformed into the RML mappings, later used to transform the invoice XML file into an RDF graph.

Our YARRRML mapping contains six mappings, each representing one different class of the EDIFACT ontology. Compared to the EDIFACT ontology, only six of the seven classes are displayed, as the class *InvoiceContent* only has the purpose of providing a domain or range for some resources. When creating each mapping, two types of rules must be defined: (i) rules for defining the data sources, i.e., the XML files of the EDIFACT invoices, and (ii) rules for generating the RDF triples. In total, there are 245 different data source rules, since a rule is generated for each EDIFACT element. Depending on which information is to be displayed in the mappings, the data sources are assigned to the mappings. The rules for generating the RDF triples are divided into two parts. Part one defines the rule for identifying the subject of the RDF triple. In our tool, this is done by the added attributes from the preprocessing 3.2. The second part defines the rules for the predicates and objects of the RDF triple. The predicate rule contains the ontology resources, and the object rule contains the identification of the value of the predicate. In our mapping, these values are either the values of XML elements or ontology resources. The RML mappings obtained with YARRRML and the invoice XML files are then fed into the RMLMapper[4] tool, to obtain the invoice RDF graph. Exemplary transformation of the *NAD segment* from an EDIFACT message to an RDF graph can be seen in Line 4 in Listing 1.1 to Lines 17-24 and Lines 26-28 in Listing 1.2.

[3] E.g., our YARRML file has 2,924 lines vs. 366,709 lines in the generated RML file.
[4] `https://github.com/RMLio/rmlmapper-java`

Listing 1.3. SHACL constraint for mandatory and single property values

```
:ExistenceDocumentNumber
   a sh:NodeShape;
   sh:targetClass edifact-o:InvoiceDetails;
   sh:property [
      sh:path edifact-o:hasDocumentNumber;
      sh:minCount 1;
      sh:maxCount 1; ] .
```

Listing 1.4. SHACL constraint for formatting check (datatype and length)

```
:LengthDocumentNumber
   a sh:NodeShape;
   sh:targetClass edifact-o:InvoiceDetails;
   sh:property [
      sh:path edifact-o:hasDocumentNumber;
      sh:datatype xsd:string;
      sh:maxLength 12; ] .
```

Invoice Validation using SHACL Once the invoice RDF graph has been created, the next step is validating the knowledge graphs. We use the W3C recommended language, Shapes Constraint Language (SHACL) [13], for the validation of RDF graphs. Constraints in SHACL can be specified over specific classes of the ontology using the *sh:targetClass* predicate, and over specific properties of the target class using the *sh:path* predicate. In our work, the shapes were created based on the input of the domain experts. For validating the correctness of the representation we create shapes based on the EDIFACT invoice guidelines. For example, shapes can express mandatory or conditional modules according to these guidelines. Listing 1.3 shows a SHACL constraint for specifying that the *documentNumber* property is mandatory (*sh:minCount* 1) and single valued (*sh:maxCount* 1) for entities of the class *InvoiceDetails*. SHACL can also be used to specify constraints about the formatting of the EDIFACT elements, including length, number, and permitted characters. Listing 1.4 shows a SHACL constraint for checking the datatype (*sh:datatype*) and the length (*sh:maxLength*) for the *documentNumber* property of the target class *InvoiceDetails*.

In addition to checking the structure and completeness of the invoices, EDIFACT-VAL also checks the logical correctness of the information with SHACL. This includes, for example, that the total net values of the goods in an invoice correspond to the sum of the net amounts of the items sold in the invoice. This type of constraint that involves aggregations over values of several RDF triples can be expressed in SHACL using SPARQL queries. Listing 1.5 shows how the aforementioned constraint is formulated with SHACL and SPARQL.

Listing 1.5. SHACL constraint to check that the total net value of the invoice is equal to the sum of the value of the items listed in the invoice

```
:SumNetPrice a sh:NodeShape ;
   sh:targetClass edifact-o:InvoiceDetails ;
   sh:sparql [
      a        sh:SPARQLConstraint ;
      sh:message "edifact-o:hasTotalLineItemAmount must equal the sum of all values of
           edifact-o:hasLineItemAmount";
      sh:prefixes [ sh:declare [
                 sh:prefix   "edifact-o" ;
                 sh:namespace "https://purl.org/edifact/ontology#"^^xsd:anyURI ; ] ] ;
   sh:select
   """SELECT $this (edifact-o:hasTotalLineItemAmount AS ?path) (?totalAmount AS ?value)
      WHERE { $this a edifact-o:InvoiceDetails ; edifact-o:hasTotalLineItemAmount ?
           totalAmount .
           { select $this (sum(?itemAmount) as ?sum) {
             ?item edifact-o:isItemOf ?invoice; edifact-o:hasLineItemAmount ?itemAmount .
             ?invoice edifact-o:hasInvoiceDetails $this . } group by $this }
         FILTER (?sum != ?totalAmount) }""" ; ] .
```

4 Evaluation

In this section, we empirically evaluate the developed EDIFACT ontology and EDIFACT-VAL tool in terms of soundness and performance. Concretely, we focus on the following core questions:

Q1 Does the EDIFACT ontology meet the state-of-the-art standards? (§4.2)
Q2 Are RDF graphs created with EDIFACT-VAL sound? (§4.3)
Q3 Are the validation of EDIFACT-VAL results complete and correct? (§4.4)
Q4 How long does EDIFACT-VAL take to validate an EDIFACT invoice? (§4.5)
Q5 Is EDIFACT-VAL applicable to real-world business processes? (§4.6)

4.1 Experimental Set Up

Dataset: We use 44 different real-world EDIFACT messages from 12 different suppliers and 6 different business cases to evaluate the performance of the EDIFACT-VAL tool. The selection of messages has been made together with the EDI experts from E/D/E to have a range of messages representing the day-to-day business workflow. Since each supplier may include different segments and data elements in the invoices, the selected messages represent a wide range of segments and segment combinations.

Tool Implementation: EDIFACT-VAL is implemented in Python 3. For SHACL validation, we use pySHACL [24], an open-source Python library. The EDIFACT-VAL is available online.[5] EDIFACT-VAL is equipped with 394 SHACL constraints provided by the experts following the EDFICAT standard. All experiments have been conducted on a machine with Intel Core i5 CPU and 8 GB of RAM.

4.2 Results of the Ontology Evaluation

Compliance with Ontology Best Practices We assessed our EDIFACT Ontology concerning current standards and best practices for ontology publication using OOPS! [19]. OOPS! (OntOlogy Pitfall Scanner!) is a web service[6] that receives the URI of the ontology and performs checks in the structural, functional, and usability profiling. In total, OOPS! tests for 41 pitfalls concerning modelling decisions, ontology language, ontology clarity, ontology understanding, no inference, wrong inference, application context, common sense, and requirement completeness. The OOPS! results show that our EDIFACT ontology meets state-of-the-art ontology standards. Only one pitfall (P22) occurred during the evaluation regarding naming conventions in the ontology terms. Since we incorporate resources from eleven ontologies, some have different naming conventions. For example, in the vCard Ontology [16], the term 'has' is always in front of a data property, `http://www.w3.org/2006/vcard/ns#hasStreetAddress`, whereas the Dublin Core-Ontology [2] does not do it, `http://purl.org/dc/elements`

[5] `https://github.com/DE-TUM/EDIFACT-VAL`
[6] `https://oops.linkeddata.es`

`/1.1/date`. Yet, this pitfall is marked as "minor level" in OOPS!, which means that it does not affect the overall quality of the EDIFACT Ontology.

Coverage of the Competency Questions To ensure that the EDIFACT Ontology satisfies the competency questions CQ (cf. Listing 1), we have translated each CQ into a SPARQL query. The SPARQL queries are available in a Jupyter notebook.[7] Using the ontology and one EDIFACT message from the dataset, we execute the SPARQL queries using the Python library `rdflib` [14]. The results show that the CQ can be answered with our RDF representations.

4.3 Completeness of the Invoice KG Generated by EDIFACT-Val

In this work, we define completeness as follows: An invoice RDF graph is complete if every *relevant* data element in an EDIFACT message is represented in the graph. Since relevance is a domain-specific criterion, we consulted with domain experts to categorize elements in EDIFACT as relevant or negligible. Then, we analyze the RDF graph completeness considering the three cases of how the EDIFACT data corresponds with RDF representations:

Case i (No Correspondence): The EDIFACT element is deemed negligible by experts and, therefore, is not encoded in the RDF graph.

Case ii (One-to-one Correspondence): The EDIFACT element is directly represented as one element in the EDIFACT Ontology.

Case iii (One-to-many Correspondence): The EDIFACT element is represented using several elements of the EDIFACT Ontology.

Distinguishing between these cases allows us to better understand and quantify the completeness of the generated RDF graphs. They ensure that no important information in the EDIFACT messages is omitted or incorrectly translated into the RDF representations.

Case i indicates that the negligible EDIFACT elements should not affect the KG completeness. In our dataset, we identified 44 of 438 of these elements.

Case ii allows for a straightforward measurement of completeness by computing the ratio between the number of RDF triples – for which the terms have a one-to-one correspondence with EDIFACT – and the number of these EDIFACT data items in the messages. After this evaluation, we obtained that the RDF graphs produced by edifact-val are complete (i.e., completeness 1.0).

Case iii entails a more sophisticated transformation (compared to the previous case) from the EDIFACT format into the RDF representation. The one-to-many correspondence in EDIFACT elements occurs in the representation of organizations, where EDIFACT combines the qualifier and identification of the organization in a single element. Yet, these are represented using several (more fine-grained) properties in the EDIFACT Ontology as they model different pieces of information about the organizations. The one-to-many correspondence case

[7] `https://github.com/DE-TUM/edifact-ontology/blob/main/CompetencyQuestions/CQ-SPARQL.ipynb`

Table 3. Validation results of SHACL constraints for different business processes (BP). BP names are omitted in accordance with privacy regulations

Business Process	**BP1**	**BP2**	**BP3**	**BP4**	**BP5**	**BP6**	**Total**
EDIFACT Invoices	96	9	7	12	906	8	1,038
SHACL Constraints	86	81	48	36	57	86	394
Violations	692	41	45	58	859	38	1,733

is handled by EDIFACT-VAL in the invoice pre-processing step (cf. Sect. 3.2). In new qualifiers can be defined during production by the users, EDIFACT-VAL implements a built-in mechanism to display unknown qualifiers for elements. In all tested 1,038 EDIFACT invoices, only 2 invoices yield 'unknown qualifier', i.e., the completeness of the RDF graph is 0.998 (out of 1.0). We inspected these invoices and found that these cases only occur due to non-standardized and self-created qualifiers in the original EDIFACT invoice. Based on these results, we can conclude that all admissible qualifiers are included in the resulting RDF graphs. I.e., EDIFACT-VAL achieves 1.0 completeness for admissible qualifiers.

4.4 Results of Invoice Validation with EDIFACT-Val

First, as a controlled evaluation, we manually introduced errors in a sample of EDIFACT invoices to test whether EDIFACT-VAL can detect them. These errors would cause violations of the SHACL constraints defined in our approach. These results show that EDIFACT-VAL successfully detected all the synthetic errors.

Next, we perform the validation over the entire dataset of EDIFACT messages. For this analysis, we present the results of the SHACL validations over the constraints defined for each business process in E/D/E. Table 3 presents the results of this evaluation. The results show the original EDIFACT invoices indeed contain errors that may affect the correctness of the invoice and compromise the integrity of related business processes.

4.5 Runtime Performance of EDIFACT-Val

We measure the efficiency of the EDIFACT-VAL when processing the invoices, i.e., the elapsed time between the tool receiving an EDIFACT message and producing the validation result. This time includes the generation of the RDF graph and its evaluation using the SHACL constraints. We selected one EDIFACT message per supplier, which may contain several invoices with varying numbers of EDIFACT data items. We ran EDIFACT-VAL 10 times on each EDIFACT message using the `hyperfine` [18] command-line benchmarking tool. Figure 3 reports the average and variance of the runtime per message. These results indicate that the EDIFACT-VAL runtime is impacted by the number of elements in the messages, as expected. Overall, the mean time for validating an EDIFACT message is within an interval of 10 to 12 seconds. Only EDIFACT messages with more than 5,000 data items require more processing time, around 30 seconds.

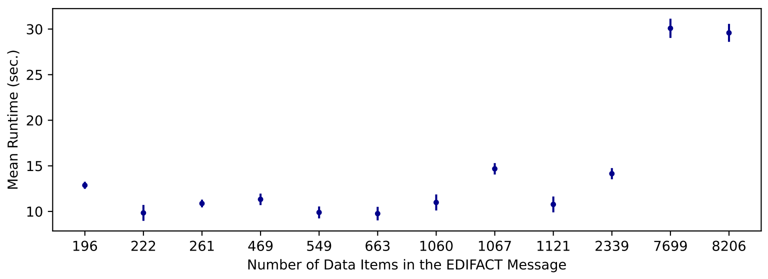

Fig. 3. EDIFACT-VAL runtime (sec.) for processing an EDIFACT message

4.6 In-Use Evaluation

Since the development was motivated by the group purchasing organization E/D/E, we also evaluated the applicability of our proposed solution approach to the E/D/E invoice processes. We asked the domain experts to validate the EDIFACT messages following the usual procedure, which is done manually. On average, it takes the experts around 30 minutes to validate one EDIFACT message. In comparison to EDIFACT-VAL (cf. Figure 3), the longest runtime is in the order of 30 seconds, which results in a time saving of around 98%.

We also asked the domain experts to analyse the violations reported by EDIFACT-VAL using the SHACL constraints. We learned that while the violations correctly capture the strictly formulated EDIFACT guidelines, certain violations are not critical in practice. This information can be included in the SHACL constraints implemented by EDIFACT-VAL using the *sh:severity* predicate of SHACL. This will be part of our future work.

5 Related Work

Ontologies for Electronic Invoices Schulze et al. [21] proposes an ontology, the Purchase-To-Pay Ontology (P2P-O) to represent electronic invoices. P2P-O relies on the European Standard EN 16931-1:2017 intending to provide ontology resources for all mandatory information in a purchase-to-pay process, which includes the invoicing process. In contrast, we aim not only to provide mandatory information for processing an electronic invoice, we also aim to validate the completeness and syntactical correctness of the invoices regarding their specific format. Furthermore, the terms used in the EDIFACT messages are not captured in the P2P-O ontology. This is why the creation of a standard-specific ontology, the EDIFACT ontology, is crucial in this case.

In the literature, we also found several ontologies and vocabularies that provide invoice-related definitions. The ones we reuse can be found in Table 2. However, they only present small parts of invoices. For example, schema.org [9] provides terms for the amount of money and currencies. Yet, these vocabularies are not specific enough to model all the EDIFACT terms.

Invoice Validation Several works have tackled the problem of invoice validation from different perspectives. Emmanuel and Thakur [5] present an approach that implements an EDI invoice validation framework; this work focuses on checking the completeness of invoices (defined as the number of fields) by comparing actual values to expected values as given in an XML file. The expected value is defined in a rule set. Similarly, our work validates invoices but uses KG technologies and more expressive SHACL constraints formulated by experts and the EDIFACT standard. Other approaches also perform invoice validation but not in the context of EDI. For example, Sál [20] presents a tool to check whether the invoice fields are provided correctly to a digital system w.r.t. to the original invoice in PDF. Other solutions [1, 3, 10] propose processing and classifying invoices in PDF using Machine Learning (ML) models. All these works check for the correctness of the translation of the invoices into machine-readable formats, and not the correctness of the original invoices. These approaches greatly differ from EDIFACT-VAL, as it processes invoices that are already in a machine-readable format and validates the correctness of the original EDIFACT invoice. Lastly, the work by Schulze et al. [21] uses SPARQL queries to perform analytical tasks on the invoices. Examples of these include finding items that are sold in large cities. In contrast, EDIFACT-VAL is tailored to check the conformance of the invoices to the EDIFACT standard and other constraints defined by experts.

6 Conclusion and Future Work

We presented a novel automatic approach EDIFACT-VAL that assists EDI experts in validating EDIFACT messages with the help of an invoice knowledge graph. The EDIFACT messages are transformed into RDF graphs using the proposed EDIFACT Ontology and RML mappings. The graphs are further validated using SHACL constraints specified with the input from domain experts.

Our experiments show that our proposed solutions enable the validation of EDIFACT invoices effectively. In particular, the evaluation with domain experts revealed that EDIFACT-VAL can considerably reduce the manual efforts. One crucial takeaway from this evaluation is that the benefit of automatic approaches like EDIFACT-VAL relies on the quality of the validation constraints. Creating too strict constraints can result in mistakenly flagging invoices as faulty even though the errors may not affect the invoice processing workflow. Therefore, assigning proper severity levels to the constraints is essential to develop usable solutions.

Future work can extend our solution for different business documents in different EDIFACT formats, i.e. purchase order message (ORDERS) or despatch advice message (DESADV). Furthermore, the obligatory implementation of electronic invoices in Germany by 2025 will make the application of open standards more prominent, especially for organizations that find the cost or time required for EDIFACT implementation too demanding. In this line, we hope that our work contributes to the implementation of electronic invoice processing using open knowledge graph and semantic web technologies.

References

1. Baviskar, D., Ahirrao, S., Kotecha, K.: Multi-layout unstructured invoice documents dataset: A dataset for template-free invoice processing and its evaluation using ai approaches. IEEE Access **9**, 101494–101512 (2021). https://doi.org/10.1109/ACCESS.2021.3096739
2. DCMI Usage Board: DCMI Metadata Terms. `https://www.dublincore.org/specifications/dublin-core/dcmi-terms/` (2020)
3. Desai, D., Jain, A., Naik, D., Panchal, N., Sawant, D.: Invoice processing using rpa & ai. In: Proceedings of the International Conference on Smart Data Intelligence (ICSMDI 2021) (2021)
4. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: Rml: A generic language for integrated rdf mappings of heterogeneous data. Ldow **1184** (2014)
5. Emmanuel, M., Thakur, S.: An approach to develop invoice validation framework. International Journal of Engineering Research & Technology (IJERT) **1** (2012)
6. EN16931-1:2017: Electronic invoicing-part 1 : Semantic data model of the core elements of an electronic invoice. Tech. rep., European Committee for Standardization (2017)
7. García-González, H., Boneva, I., Staworko, S., Labra-Gayo, J.E., Lovelle, J.M.C.: Shexml: improving the usability of heterogeneous data mapping languages for first-time users. PeerJ Computer Science **6**, e318 (2020)
8. Grüninger, M., Fox, M.S.: The role of competency questions in enterprise engineering. In: Benchmarking—Theory and practice, pp. 22–31. Springer (1995)
9. Guha, R.V., Brickley, D., Macbeth, S.: Schema.org: Evolution of Structured Data on the Web. Communications of the ACM **59**(2), 44–51 (2016)
10. Gunaratne, H., Pappel, I.: Enhancement of the e-invoicing systems by increasing the efficiency of workflows via disruptive technologies. In: Electronic Governance and Open Society: Challenges in Eurasia: 7th International Conference, EGOSE 2020, St. Petersburg, Russia, November 18–19, 2020, Proceedings 7. pp. 60–74. Springer (2020)
11. Horridge, M., Gonçalves, R.S., Nyulas, C.I., Tudorache, T., Musen, M.A.: Webprotégé: A cloud-based ontology editor. In: Companion Proceedings of The 2019 World Wide Web Conference. pp. 686–689 (2019)
12. Iglesias-Molina, A., Chaves-Fraga, D., Dasoulas, I., Dimou, A.: Human-Friendly RDF Graph Construction: Which One Do You Chose? In: International Conference on Web Engineering. pp. 262–277. Springer (2023)
13. Knublauch, H., Kontokostas, D.: Shapes constraint language (SHACL). W3C recommendation, W3C (Jul 2017), https://www.w3.org/TR/2017/REC-shacl-20170720/
14. Krech, D., Grimnes, G.A., Higgins, G., Hees, J., Aucamp, I., Lindström, N., Arndt, N., Sommer, A., Chuc, E., Herman, I., Nelson, A., McCusker, J., Gillespie, T., Kluyver, T., Ludwig, F., Champin, P.A., Watts, M., Holzer, U., Summers, E., Morriss, W., Winston, D., Perttula, D., Kovacevic, F., Chateauneu, R., Solbrig, H., Cogrel, B., Stuart, V.: RDFLib (Aug 2023). https://doi.org/10.5281/zenodo.6845245, `https://github.com/RDFLib/rdflib`
15. Lohmann, S., Link, V., Marbach, E., Negru, S.: Webvowl: Web-based visualization of ontologies. In: Knowledge Engineering and Knowledge Management: EKAW 2014 Satellite Events, VISUAL, EKM1, and ARCOE-Logic, Linköping, Sweden, November 24-28, 2014. Revised Selected Papers. 19. pp. 154–158. Springer (2015)

16. McKinney, J., Iannella, R.: vCard Ontology - for describing People and Organizations. W3C note, W3C (May 2014), https://www.w3.org/TR/2014/NOTE-vcard-rdf-20140522/
17. Noy, N.F., McGuinness, D.L., et al.: Ontology development 101: A guide to creating your first ontology (2001)
18. Peter, D.: hyperfine (Mar 2023), `https://github.com/sharkdp/hyperfine`
19. Poveda-Villalón, M., Gómez-Pérez, A., Suárez-Figueroa, M.C.: Oops!(ontology pitfall scanner!): An on-line tool for ontology evaluation. International Journal on Semantic Web and Information Systems (IJSWIS) **10**(2), 7–34 (2014)
20. Sál, J.: Data mining as tool for invoices validation. IT for Practice 2018 p. 121 (2018)
21. Schulze, M., Schröder, M., Jilek, C., Albers, T., Maus, H., Dengel, A.: P2p-o: A purchase-to-pay ontology for enabling semantic invoices. In: The Semantic Web: 18th International Conference, ESWC 2021, Virtual Event, June 6–10, 2021, Proceedings 18. pp. 647–663. Springer (2021)
22. Shimizu, C., Hirt, Q., Hitzler, P.: Modl: A modular ontology design library. arXiv preprint arXiv:1904.05405 (2019)
23. Shotton, D.: FRAPO, the Funding, Research Administration and Projects Ontology. Tech. rep. (april 2017), http://purl.org/cerif/frapo
24. Sommer, A., Car, N.: pySHACL (Jan 2022). https://doi.org/10.5281/zenodo.4750840, `https://github.com/RDFLib/pySHACL`
25. Suárez-Figueroa, M.C., Gómez-Pérez, A., Fernández-López, M.: The neon methodology for ontology engineering. In: Ontology engineering in a networked world, pp. 9–34. Springer (2011)
26. Van Assche, D., Delva, T., Heyvaert, P., De Meester, B., Dimou, A.: Towards a more human-friendly knowledge graph generation & publication. In: ISWC2021, The International Semantic Web Conference. vol. 2980. CEUR (2021)