Davan Chiem Dao, Christophe Debruyne, and Paul Stijfhals. Using knowledge graphs and SHACL to validate declaration forms: An experiment in the social security domain to assess shacl's applicability. In European Commission, Directorate-General for Informatics, and Publications Office of the European Union, editors, ENDORSE, 2nd EuropeaN Data conference On Reference data and SEmantics ENDORSE 2023 - Virtual event, 14-16 March 2023 - Proceedings, pages 85-96. Publications Office of the European Union, 2023

Using Knowledge Graphs and SHACL to Validate Declaration Forms: An Experiment in the Social Security Domain to Assess SHACL's Applicability

Davan Chiem Dao
Montefiore Institute
University of Liège
4000 Liège
Belgium

Christophe Debruyne[1]
Montefiore Institute
University of Liège
4000 Liège
Belgium

Paul Stijfhals
Smals Research
Smals asbl
1060 Brussels
Belgium

Abstract

Smals carries out innovative ICT projects in e-government and e-health in Belgium. Smals Research conducts applied research into novel technologies to see whether they can improve or provide new solutions to their clients. In 2021, these technologies included knowledge graphs and SHACL. In this paper, we report on a study we conducted in 2021 at Smals Research and continued in 2022 at the University of Liège. Many use cases within Smals' clients rely on XML for various reporting purposes, validated with a combination of XML Schema Definition (XSD) documents and bespoke code. We wanted to know to what extent SHACL is more expressive than XSD and to what extent SHACL can scale to support such data validation tasks. To answer this question, Smals Research has conducted an experiment in which we validate declaration forms in the Belgian social security domain. Our study indicates that while SHACL is more expressive than XSD for these declaration forms, how SHACL shapes are declared profoundly impacts reusability and efficiency.

1. Introduction

The Shapes Constraint Language (SHACL) (Knublauch & Kontokostas, 2017) allows us to validate RDF (Schreiber & Raimond, 2014) graphs by declaring constraints for RDF resources that are called *shapes*. These shapes describe the characteristics that RDF resources and their properties should possess. These shapes can define conditions on, among others, the presence or absence of certain properties and property paths, the relationship between a the value of a property and the value of property paths, the allowed data types of properties, and the cardinality of properties.

In public administrations and the EU, SHACL has been used to implement constraints put forward by so-called application profiles. For example, the Flemish government's Open Standaarden voor Linkende Organisaties (OSLO) provides JSON-LD (Kellogg, Longley, & Champin, 2020) contexts as schemas for data exchange, which can be validated against corresponding SHACL templates. These profiles are used to validate data and are shared with the public. That said, the shapes described in those application profiles are often limited to "simple" integrity constraints such as cardinality, length, and regular expressions. Complex rules or constraints must be validated in industrial and organizational knowledge graphs (KGs) (Hogan et al., 2020). Such constraints, which we will now call *complex constraints*, are often outside the scope of the basic constructs provided by SHACL. Luckily, those constraints can be declared by "implementing" those constraints using SPARQL (Seaborne & Harris, 2013), which is the RDF query language. Those SPARQL queries are embedded in so-called SPARQL Constraint Components, which does require knowledge of yet another technology.

[1] Corresponding author: Christophe Debruyne – c.debruyne@uliege.be

Smals[2] carries out innovative ICT projects in e-government and e-health for social security and healthcare institutions. Smals Research[3] conducts applied research into novel technologies to see whether they can improve or provide new solutions to their clients. In 2021, these technologies included RDF KGs and SHACL to validate the data in KGs

Many use cases within Smals' clients rely on XML for various reporting purposes, validated with a combination of XML Schema Definition (XSD) documents and bespoke code. The bespoke code was used to validate constraints that were too complex for XSD. We wanted to know to what extent SHACL can scale to validate forms against shapes that implement complex constraints, which need to rely on a combination of existing SHACL constructs, logical operations, and embedded SPARQL queries. To answer this question, Smals Research has conducted an experiment in which we validate declaration forms in the Belgian social security domain (RSZ-ONSS). This study was conducted in 2021 at Smals Research and continued in 2022 at the University of Liège. We report on this study and our findings in this paper.

The paper is organized as follows: Section 2 describes the context of the Belgian social security and the DmfA declaration that is the focus of our study. Section 3 reports on our method for constructing an RDF knowledge graph, transforming DmfA declarations into RDF and developing SHACL shapes. Section 4 provides an overview of recurring patterns for which one could foresee abstractions and some design considerations we took to heart while developing the shapes. Section 5 reports on a demonstration, and Section 6 presents an overview of the conclusions and lessons learned. Section 6 also prevents an overview of possible venues for future work.

1.1 Disclaimer

While Smals builds solutions for the RSZ-ONSS, this study was not conducted for the RSZ-ONSS. This study was conducted to gain insights into the opportunities offered by SHACL. We chose this domain as it was sufficiently representative concerning complex constraints and the online availability of all information documentation and files.

2. Context: The Belgian Social Security and DmfA Declarations

The RSZ-ONSS[4] is Belgium's national security office and manages Belgium's social security system. As such, the RSZ-ONSS is responsible for collecting social security contributions from employers and employees and for distributing these contributions to the various social security funds (e.g., healthcare, pensions, and unemployment benefits).

To reduce the number of filled-out forms and interactions between RSZ-ONSS and employers or employees and simplify forms to speed up the process, the RSZ-ONSS developed three electronic declarations as part of their e-government program. Employers use these forms to send information about their employers to the RSZ-ONSS.

- The *Déclaration immédiate/onmiddelijke aangifte*, which translates to "immediate declaration" (Dimona), contains information concerning the start and end of an employment relationship.
- The *Déclaration des Risques Sociaux* (DRS), which translates to "declaration of social risks," is used to declare that an employee encountered a social risk during the

[2] https://smals.be/
[3] https://www.smalsresearch.be/
[4] Rijksdienst voor Sociale Zekerheid - Office National de Sécurité Sociale: https://www.socialsecurity.be/

employment relationship. Social risks are events that change an employee's social position, such as being fired, being a victim of an accident at work, or suffering an illness for an extended period.

- The *Déclaration multifonctionnelle/multifunctionele Aangifte* (DmfA), which translates to "multifunctional declaration," is used to communicate more general information about an employment relationship. The various branches of the RSZ-ONSS require that information.

One could represent these declarations on a timeline where a Dimona declaration provides information on the start- and end date of an employment relationship, a DfmA declaration provides period information about that employment relationship, and a DRS declaration provides information on events during those periods in the employment relationship.

In this study, we wanted to assess the applicability of SHACL in this e-government process. We focused on DmfA declarations as it is the most complex electronic declaration; it has the most fields subject to complex constraints. Thus, it is the most suitable declaration to reveal to what extent KGs and their technologies can potentially improve processes supported by the existing information system.

2.1 DmfA Declarations

The DmfA is a multifunctional declaration that employers send to the RSZ-ONSS. It is multifunctional because all institutions use it for social security purposes, e.g., determining the amount of contribution an employer owes and allocating social rights indemnity payments.

Concerning the kind of information transmitted, it includes salary data and the employee's working time. It can be sent via the Web by manually filling in an online form or via file transfer, for which the information is contained in an XML file. The Web interface covers many data validation aspects, though filling those in manually can be tedious for companies employing multiple people. XML is used for communicating information about multiple employment relationships in a batch. The Belgian social security has made the XSD schema and a simple Java application available for some "superficial" data validation. This latter format for a DmfA will be the input source of the validation process. Most employers report using XML, and the validation of those XML documents will be the focus of our study.

A glossary[5] provided by the RSZ-ONSS documents the content of the XML file. The XML file's structure and schema are described using an Entity-Relationship Diagram (ERD); the ERD contains 29 entity sets, 28 relationship sets, and over 200 attributes (across the entity sets). The ERD also indicates which attributes are prohibited. The fact that some entities, relationships, or attributes can become prohibited highlights that the DmfA changes over time. Indeed, as the regulations change, the DmfA must reflect these changes. Hence, there are different versions, each corresponding to a particular quarter. When an employer sends a DmfA for a particular quarter, it must respect the constraints of that quarter. This temporal aspect should be taken into consideration when designing a validation process. It is important to note that the glossary often refers to so-called appendices, which contain additional (structured) information that needs to be consulted to validate the data. An example would be the time interval a particular code was valid. Those appendices also evolve over time.

To show how constraints are described in the glossary, an example is provided in Figure 1. It illustrates the glossary's HTML version, but PDF and XML versions also exist. One can

[5] https://www.socialsecurity.be/lambda/portail/glossaires/dmfa.nsf/web/glossary_home_fr

observe that some constraints are structured, such as the compulsory presence of an attribute or the maximum length of a value, which can both be expressed in XSD. Nevertheless, most constraints are informally described. While some of these constraints can be expressed in XSD (e.g., a regular expression), many of them cannot and require to be validated by an application (e.g., computing a checksum of an ID number).

Currently, there are two validation processes put in place by the RSZ-ONSS. First, the most uncomplicated validation process consists of a lightweight Java program. This program is available to employers to verify a DmfA declaration before submission. This program verifies whether the DmfA declaration is a well-formed XML file that conforms to the XSD. Additionally, two constraints are verified: the uniqueness of the Social Security Identification Numbers and the amount owned declared corresponds to the one computed. Data validation capabilities by the employer are thus limited. The other validation process occurs upon the submission of a DmfA. The RSZ-ONSS runs a non-disclosed program that validates all the constraints a DmfA should respect. However, the validation result can take up to ten days to be sent.

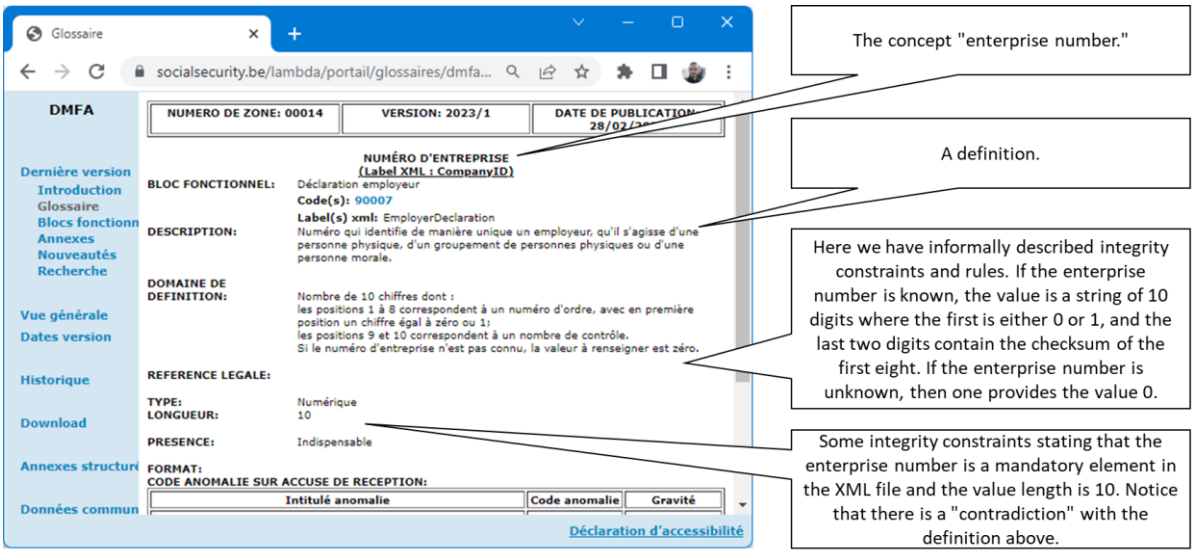


Figure 1 A page of the DmfA glossary describing enterprise numbers.

These explanations have highlighted some key issues with the current situation. Many constraints fall outside the expressivity of XSD and are described in natural language and thus cannot be processed by a computer agent or are written in a non-interoperable format (i.e., Java code or non-disclosed). The current validation processes are either partially complete or not directly available for employers, increasing the required time to fill the declaration. Knowledge graph technologies and SHACL can hopefully overcome these problems by creating interoperable data validation constraints.

3. Method

We built a prototype knowledge graph to determine to what extent SHACL can scale to validate complex constraints. The steps are relatively straightforward. We first created a vocabulary for our knowledge graph (without SHACL shapes) and then transformed the glossary and the different appendices into RDF, committing to that vocabulary. We also

created a mapping from DmfA declarations in XML to RDF. Once everything was implemented, we developed the SHACL shapes and a process for validating DmfA XML files. All of these steps will be detailed in the following subsections.[6]

3.1 Building the Vocabulary

As this knowledge graph project did not require the support for complex reasoning tasks and our Universe of Discourse was fairly simple, we decided to develop a vocabulary rather than an ontology. The development of this vocabulary followed the following steps. We first lifted the XSD schema into a vocabulary. Each functional block corresponds with a concept (or class in the vocabulary), and the relationships between concepts have been made explicit. We then lifted the schemas of the appendices into the vocabulary. Finally, we used the ERD and some domain knowledge to refine the vocabulary. One example is the homogenization of temporal aspects (using quarters throughout the knowledge graph instead of a combination of quarters and years).

3.2 Data Transformation

While integrating non-RDF data into the RDF knowledge graph is not key to this study's report, we felt it was sufficiently important to describe our experiences.

As DmfA declarations are stored as XML, and the appendices are available as XML and CSV files (among others), we were able to use declarative mapping languages such as RML (Dimou et al., 2014) to transform (and integrate) the data into an RDF knowledge graph. RML allows us to declare how data in those files should be transformed into RDF. RML mappings are stored as RDF and are part of the knowledge graph. Such an approach thus aids in answering data lineage and data provenance questions such as: "Where do the values for a particular property come from?"

One of the advantages of RML is its availability of FNO (De Meester, Maroy, Dimou, Verborgh, & Mannens, 2017), allowing us to use functions to manipulate data during the RDF generation process. Unfortunately, the FNO prototype implementation did not implement all GREL[7] functions, and we had to resort to some custom scripts in some cases.

Another limitation was that, at the time of the study, we could not manipulate the source file before RDF generation; you refer to a file as the source.[8] As the XML file is a tree, there are no cycles. The knowledge graph, on the other hand, does. The XML file contained many repeating elements of the same types, and we had to find a way to assign them internal IDs to ensure that the right RDF resources were related. Thus, we had to include those IDs with a script before generating RDF, which made the RDF generation process less self-contained. A recent proposal (Delva, Van Assche, Heyvaert, De Meester, & Dimou, 2021) may address this issue.

Of the 24 appendices, 19 were integrated. Of the remaining 5, 4 appendices were unnecessary (e.g., land codes and ASCII conversion codes), and 1 was too complex as we had no access to domain expertise.

---

[6] The results of this process are available on https://github.com/chrdebru/dmfa_pub

[7] https://openrefine.org/docs/manual/grelfunctions

[8] This is possible when generating data from relational databases as you can avail of the SQL query language (and thus its functions) to indicate what tables or views to transform.

3.3 Developing SHACL Shapes

This section covers the process of creating SHACL shapes for the DmfA. It covers how our SHACL shapes were generated. In the next section, we discuss emerging patterns and design considerations.

The shapes for classes and data properties were "bootstrapped" by processing the XSD, i.e., a script that generated SHACL shapes based on XSD constraints capturing data types, cardinality constraints, maximum length, patterns, etc. Figure 2 depicts some of these "basic" shapes. The starting point for a class's shape is a rule stating that each instance has valid data properties. Other shapes describe criteria data properties must meet, such as specified datatypes, value and length constraints, and patterns. These criteria are the SHACL equivalent of XSD constraints. Important to note is that all constraints had to be manually checked, as there are some "inconsistencies" in the glossary (see Figure 1). The length can refer to both the maximum length and the exact length. We choose to generate validation checks for the former and change those where necessary.

```
kg:EmployerDeclarationShape a sh:NodeShape ;
    rdfs:comment "Property Shape for EmployerDeclaration (90007)" ;
    sh:targetClass ont:EmployerDeclaration ;

    sh:property kg:QuarterShape;
    sh:property kg:NOSSRegistrationNbrShape;
    sh:property kg:TrusteeshipShape;
    sh:property kg:CompanyIDShape;
    sh:property kg:NetOwedAmountShape;
    sh:property kg:System5Shape;
    sh:property kg:HolidayStartingDateShape;
.
```

```
kg:QuarterShape a sh:PropertyShape;
    rdfs:comment "Property Shape for Quarter (00013)" ;
    sh:path ont:Quarter;

    sh:datatype xs:integer;
    sh:minInclusive 20031;
    sh:maxLength 5;
    sh:pattern "\\d{4}(1|2|3|4)";
.
```

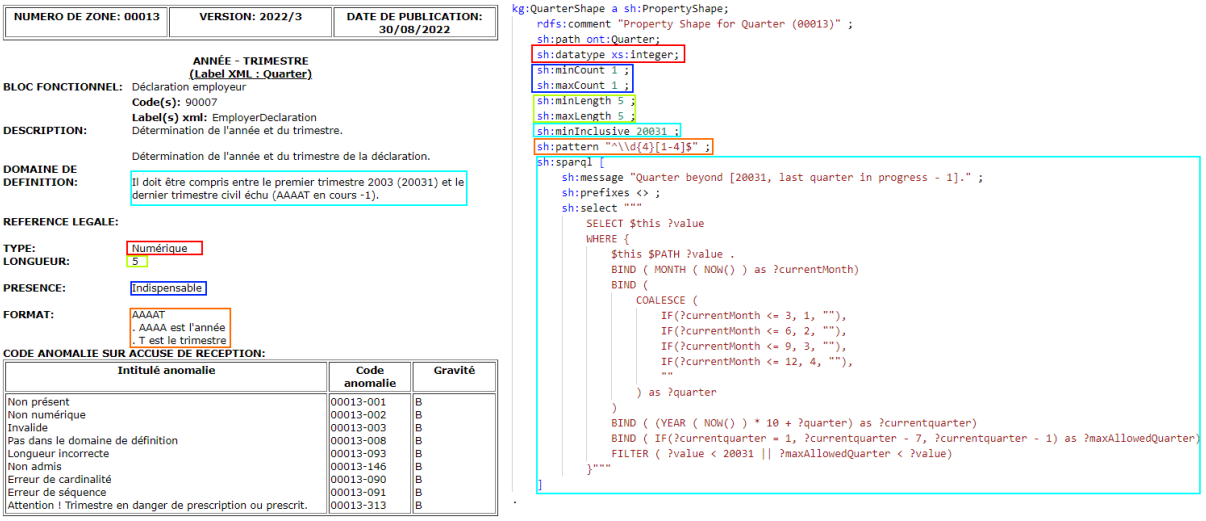
Figure 2 The DmfA shapes generated by a script as a starting point. Notice that the node shape for `ont:EmployerDeclaration` refers to various property shapes. In our figure, we illustrate the property shape for `ont:Quarter`.

These constraints do not cover all the constraints that the declaration should respect. Thus, they were manually verified and refined. The glossary of each class, data property, and object property were consulted to determine missing constraints. Missing constraints concerning classes and data properties were added to the base shapes, whereas object properties' constraints were added to the shape of their domain. Figure **3** shows how the constraints concerning `ont:Quarter` were refined (only the datatype was retained). Some of the missing constraints can be expressed with SHACL-core components (such as adding min length to simulate an exact length), while others had to be expressed with SPARQL Constraint Components. In this example, we use SPARQL to test whether "year-trimester," which contains five digits, contains a valid year (between 2003 and the time of declaration) and a valid trimester.

The example in Figure 3 contains a SPARQL query to check the validity of a property. We note that there are examples in which the graph needs to be traversed. For instance, to check whether a date of an entity falls in the interval of a related appendix. An example of such SPARQL Constraint Components will be illustrated shortly.

4 Emerging Patterns and Design Considerations

Various patterns were identified during the development of the SHACL shapes. We aim to provide constraints that can be applied in contexts beyond the social security domain or to serve as a model for creating other constraints by presenting the structure of the developed shapes. We omit some code samples due to space constraints, but examples can be found in the GitHub mentioned above repository.



NUMERO DE ZONE: 00013	VERSION: 2022/3	DATE DE PUBLICATION: 30/08/2022

ANNÉE - TRIMESTRE
(Label XML : Quarter)

BLOC FONCTIONNEL: Déclaration employeur
Code(s): 90007
Label(s) xml: EmployerDeclaration

DESCRIPTION: Détermination de l'année et du trimestre.

Détermination de l'année et du trimestre de la déclaration.

DOMAINE DE DEFINITION: Il doit être compris entre le premier trimestre 2003 (20031) et le dernier trimestre civil échu (AAAAT en cours -1).

REFERENCE LEGALE:

TYPE: Numérique
LONGUEUR: 5

PRESENCE: Indispensable

FORMAT: AAAAT
, AAAA est l'année
, T est le trimestre

CODE ANOMALIE SUR ACCUSE DE RECEPTION:

Intitulé anomalie	Code anomalie	Gravité
Non présent	00013-001	B
Non numérique	00013-002	B
Invalide	00013-003	B
Pas dans le domaine de définition	00013-008	B
Longueur incorrecte	00013-093	B
Non admis	00013-146	B
Erreur de cardinalité	00013-090	B
Erreur de séquence	00013-091	B
Attention ! Trimestre en danger de prescription ou prescrit.	00013-313	B

```
kg:QuarterShape a sh:PropertyShape;
    rdfs:comment "Property Shape for Quarter (00013)" ;
    sh:path ont:Quarter;
    sh:datatype xs:integer;
    sh:minCount 1 ;
    sh:maxCount 1 ;
    sh:minLength 5 ;
    sh:maxLength 5 ;
    sh:minInclusive 20031 ;
    sh:pattern "^\\d{4}[1-4]$" ;
    sh:sparql [
        sh:message "Quarter beyond [20031, last quarter in progress - 1]." ;
        sh:prefixes <> ;
        sh:select """
            SELECT $this ?value
            WHERE {
                $this $PATH ?value .
                BIND ( MONTH ( NOW() ) as ?currentMonth)
                BIND (
                    COALESCE (
                        IF(?currentMonth <= 3, 1, ""),
                        IF(?currentMonth <= 6, 2, ""),
                        IF(?currentMonth <= 9, 3, ""),
                        IF(?currentMonth <= 12, 4, ""),
                        ""
                    ) as ?quarter
                )
                BIND ( (YEAR ( NOW() ) * 10 + ?quarter) as ?currentquarter)
                BIND ( IF(?currentquarter = 1, ?currentquarter - 7, ?currentquarter - 1) as ?maxAllowedQuarter)
                FILTER ( ?value < 20031 || ?maxAllowedQuarter < ?value)
            }"""
    ]
```

Figure 3: Glossary of Quarter (left) and refined shape of Quarter (right)

4.1 Checksums

Checksums can be used to verify the correctness of identifiers (e.g., company IDs). Two checksums are used in DmfA (anecdotally, a reference to one of the older algorithms was missing). Each checksum can be declared and implemented in SPARQL Constraint Components, which are reused by various property shapes.

4.2 Existing Code w.r.t Appendices

Some appendices define values that certain fields in the XML can accept. Listing 1 illustrates the validity checking for `ont:PositionCode` with a set of allowed values defined by Appendix 9. This rule checks if at least one resource of type `an9:PositionCode` with a code equal to the property value that corresponds with that field. The main advantage of this rule is that it is up to date with the current state of Appendix 9.

```
[
    sh:message "Invalid code for a position, code does not exist" ;
    sh:prefixes <> ;
    sh:select """
        SELECT $this ?value
        WHERE {
            $this ont:PositionCode ?value.
            OPTIONAL{
                ?pc a an9:PositionCode ; an9:Code ?value.
            }
            FILTER(!BOUND(?pc))
        }""" ;
] .
```

Listing 1 SPARQL Constraint Component for validating values with respect to appendices.

4.3 Code Within Valid Period w.r.t Appendices

The appendices defining values for certain fields in XML documents also define their validity period. As mentioned earlier, we homogenized temporal aspects in the vocabulary; data properties concerning the starting and ending quarter of the validity of a code were added to the vocabulary so that appendices had dates. This addition made the constraints checking the temporal validity of a code follow the same pattern. It also eases the checking as the quarter of declaration must not be transformed into a date before being compared to the validity

period. An example of this type of rule is presented in Listing 2. The quarter of the declaration is reached through the inverse path from a resource and compared to the starting and ending quarter of the code that matches the value of the `ont:PositionCode` data property. Like the previous rule, being up to date with the current state of the appendix is the main advantage of this rule. This particular example also demonstrates that we do traverse graphs with SPARQL.

```
[
    sh:message "Invalid ont:PositionCode, code is out of valid quarter range." ;
    sh:prefixes <> ;
    sh:select """
        SELECT $this ?value
        WHERE {
            $this ont:PositionCode ?value.
            OPTIONAL {
                ?pc a an9:PositionCode;
                    an9:Code ?value;
                    an9:validFromQuarter ?startQuarter;
                    an9:validToQuarter ?endQuarter .
                $this ^ont:R_90012_90015/
                      ^ont:R_90017_90012/
                      ^ont:R_90007_90017/
                      ont:Quarter ?quarter.
                FILTER( ?startQuarter < ?quarter && ?quarter < ?endQuarter)
            }
            FILTER(!BOUND(?pc))
        }""" ;
] .
```

Listing 2 SPARQL Constraint Component for a code's temporal validity.

4.4 Unique Occurrences of Information Inside a DmfA Declaration

Some constraints described in the glossary state that some entity values must be unique inside a DmfA declaration. Listing 3 illustrates a rule expressing the uniqueness of natural person sequence numbers in an employer declaration. This rule counts the number of occurrences for a sequence number. The sequence number is not unique if this number is greater than one. In this example, we want to avoid two pieces of information about the same person. This pattern also appears for combinations of properties, though we do not have the space to provide an example.

```
[
    sh:message
      "Each ont:NaturalPersonSequenceNbr must be unique for a ont:EmployerDeclaration." ;
    sh:prefixes <> ;
    sh:select """
        SELECT $this
        WHERE {
            {
                SELECT $this (COUNT(?seqNbr) as ?seqNbrOcc)
                WHERE {
                    $this ont:R_90007_90017/ont:NaturalPersonSequenceNbr ?seqNbr .
                }
                GROUP BY ?seqNbr $this
            }
            FILTER(?seqNbrOcc > 1)
        }""" ;
] .
```

Listing 3 SPARQL Constraint Component for ensuring that DmfA declarations do not have multiple entities for the same person.

4.5 Design Considerations

During the implementation of SHACL shapes, various design choices were made. This section will present these choices and the considerations that led to their implementation. We will also compare the trade-offs and potential impacts of each alternative on the effectiveness and efficiency of the SHACL shapes. These considerations should be considered as they can significantly impact the usability and maintenance of the shapes. For this study, we used the TopBraid SHACL API.[9]

4.5.1 SPARQL Constraint Components Optimization

Developing SPARQL Constraint Components requires thorough query language and computer science knowledge. For instance, a first implementation of a checksum splits the number using type casting and the substring function. However, string manipulation is often slower than operating on integers. Thus, this query was improved by performing a modulo operation to split the number. On average, a reduction of 300 milliseconds was achieved.

Another example was the validation of codes where OPTIONAL was combined with a FILTER on unbound variables to speed things up. In other words, rather than intuitively following the description of the constraint, it is worthwhile to approach the problem from another angle. Here, tests showed an average reduction of 250 milliseconds.

4.5.2 Selecting the Target of a Rule

SHACL shapes were bootstrapped from the XSD schema and refined using the glossary. However, strict adherence to this approach may significantly slow the validation process. It is possible to use SPARQL queries to navigate the graph and express constraints from any resource because the declaration's graph does not have isolated vertices. Thus, rewriting a constraint to target another class may be more efficient.

An example of this is testing the uniqueness of people in a declaration. The rule that no two people with the same sequence number can occur in a declaration was described in NaturalPersonSequenceNbr; following this approach would result in a SPARQL query for each entity of that type. Declaring (and rewriting) this constraint at the parent `ont:EmployerDeclaration` is more efficient.

4.5.3 Redundancy, and Clarity vs. Efficiency

A shape's constraints may have overlapping checks. For example, consider the shape of an `ont:Quarter` shown in Figure 3. If the length constraints are not met, the pattern constraint also fails. Similarly, if the value is less than 20031, the minimum value constraint and the SPARQL Constraint Component will report errors. While the length and minimum value constraints are redundant, they offer a more specific reason for failure. An error indicating a violation of the minimum length reflects a missing character, whereas a mismatching pattern error does not. For this reason, redundant but more precise constraints were favored. Nevertheless, this choice decreases efficiency as more checks must be performed.

So, when developing constraints and constraint components, expressing those in a clear and easily understandable manner was favored other making them as efficient as possible. Clear constraints often result from a combination of simpler constraints, making them easier to maintain. As they are fragmented, constraints can be reused for other purposes. On the other hand, efficient constraints are more likely to be complex and harder to maintain.

---

[9] https://github.com/TopQuadrant/shacl

4.5.4 Knowledge Organization

One of the limitations of this study is investigating the use of named graphs for storing the evolution of annexes and appendices over time. We have not stored DmfA (and other) declarations in their own named graphs either. The former would not be that interesting of an exercise, and the latter requires sufficiently representative data to assess the impact of named graphs. That said, it is known that partitioning the RDF knowledge graph into named graphs has a positive impact on query efficiency.

5 Demonstration

We tested our SHACL shapes on some examples provided by the glossary. We also changed some examples so that they contained errors. All examples used artificial data; an arguable limitation of this study is that it was not applied to actual data. However, as stated before, we chose this as an application domain for its availability of data, documentation, and complex constraints. In Figure 4, we demonstrate the validation of a transformed DmfA declaration containing an erroneous company ID.

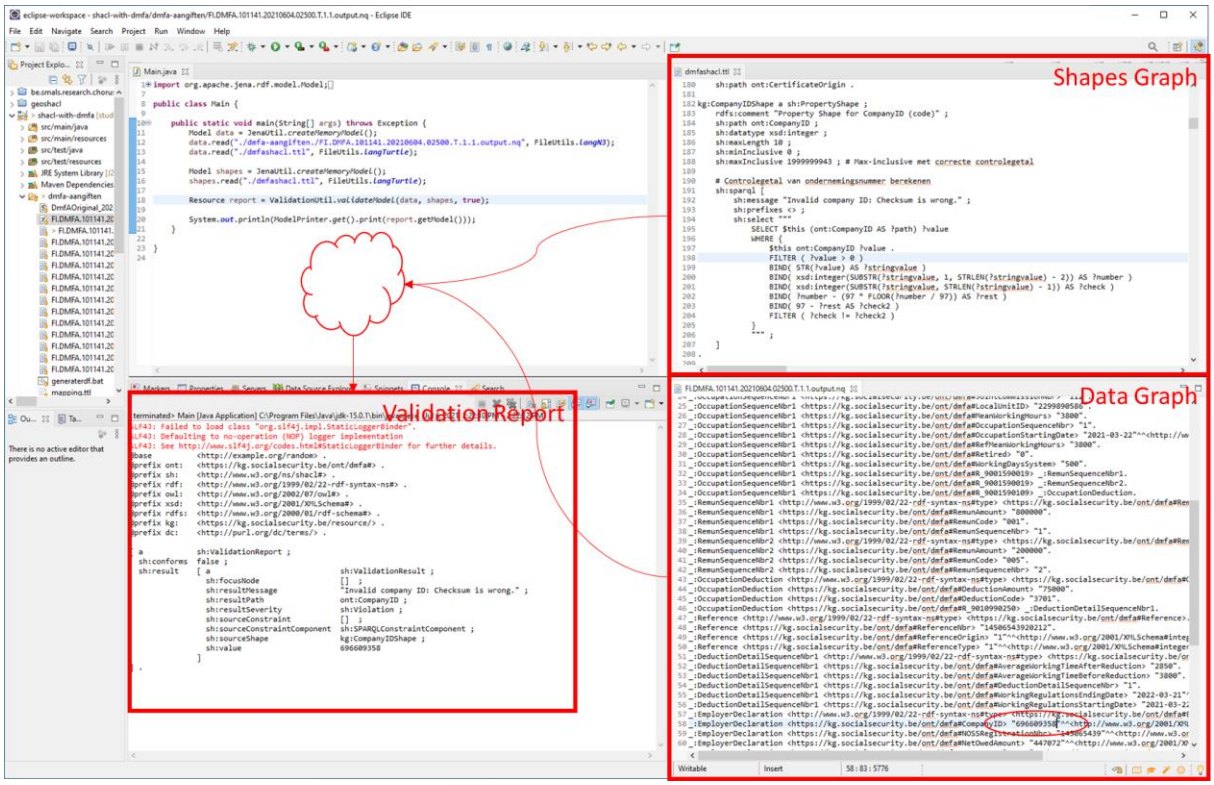


Figure 4 Validating DmfA examples provided by RSZ-ONSS in which we introduced errors. Here, it reports on an invalid checksum of a company ID.

6 Conclusions, Lessons Learned, and Future Work

We wanted to know to what extent SHACL is more expressive than XSD and to what extent SHACL can scale to support such data validation tasks. We can conclude that we can represent more validation constraints with SHACL than with XML and XSD. We can potentially share those SHACL shapes for more "thorough" DmfA data validation by an employer. I.e., we showed that the subset of the shapes one can share constitutes an application profile richer than what is possible with current XSD schemas. Our results demonstrate that this approach is viable for validating declarations against complex constraints. We know that implementing constraints that validate a declaration considering

previously submitted declarations is possible with knowledge graph technologies. However, they must be investigated in more depth in the future as they require access to the knowledge graph. A Web service can be conceived, but such a service should ensure that the knowledge graph cannot be exploited for malicious intents.

Moreover, the SHACL shapes we have developed are also more interoperable. Not only because they are part of the knowledge graph but also because SHACL processors exist for different software ecosystems (Python, Java, …). As exemplified by (Debruyne & McGlinn, 2021), one can use Linked Data (Bizer, Heath, & Berners-Lee, 2009) principles to share those constraint components within and across organizations. This opportunity was not explored in this study and is considered for future work.

Important to note is the trade-off between efficiency and clarity. We have chosen to keep the constraint components as simple (and, therefore, also as reusable) as possible, but this is at the cost of computational overhead in terms of the number of SPARQL queries being fired.

Unsurprisingly, creating SHACL shapes and constraint components is a complex knowledge engineering task. It requires knowledge of RDF and SHACL and a profound knowledge of SPARQL to implement complex constraints. Not to be underestimated is the ability to reformulate constraints to reduce overhead or increase efficiency. A prime example in this study was a constraint on entities that could be rewritten from the perspective of another related entity.

As for future work, the following elements are considered. We have already mentioned the interoperability of SHACL shapes and constraint components across software ecosystems using Linked Data principles. Other aspects pertaining to named graphs. First is the validation of declarations (all types) over time. For instance, validate revisions of a declaration, for which appropriate knowledge organization strategies need to be developed—secondly, the development and deployment of validation across declarations, employers, and employees. As stated previously, a limitation was the use of artificial examples available online to demonstrate a point. We believe we have shown said point, but assessing its impact in an operational environment would be interesting. This will be difficult due to the sensitive nature of the data and requires collaboration with the RSZ-ONSS.

References

Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*. https://doi.org/10.4018/jswis.2009081901

De Meester, B., Maroy, W., Dimou, A., Verborgh, R., & Mannens, E. (2017). RML and FnO: Shaping DBpedia Declaratively. In E. Blomqvist, K. Hose, H. Paulheim, A. Lawrynowicz, F. Ciravegna, & O. Hartig (Eds.), *The Semantic Web: ESWC 2017 Satellite Events - ESWC 2017 Satellite Events, Portorož, Slovenia, May 28 - June 1, 2017, Revised Selected Papers* (pp. 172–177). https://doi.org/10.1007/978-3-319-70407-4_32

Debruyne, C., & McGlinn, K. (2021). Reusable SHACL Constraint Components for Validating Geospatial Linked Data. In B. Yaman, M. A. Sherif, A.-C. N. Ngomo, & A. Haller (Eds.), *Proceedings of the 4th International Workshop on Geospatial Linked Data (GeoLD) Co-located with the 18th Extended Semantic Web Conference (ESWC 2021), Virtual event (instead of Hersonissos, Greece), June 7th, 2021* (pp. 59–66). Retrieved from https://ceur-ws.org/Vol-2977/paper8.pdf

Delva, T., Van Assche, D., Heyvaert, P., De Meester, B., & Dimou, A. (2021). Integrating Nested Data into Knowledge Graphs with RML Fields. In D. Chaves-Fraga, A. Dimou, P. Heyvaert, F. Priyatna, & J. F. Sequeda (Eds.), *Proceedings of the 2nd International Workshop on Knowledge Graph Construction co-located with 18th Extended Semantic Web Conference (ESWC 2021), Online, June 6, 2021*. Retrieved from https://ceur-ws.org/Vol-2873/paper9.pdf

Dimou, A., Sande, M. Vander, Colpaert, P., Verborgh, R., Mannens, E., & de Walle, R. Van. (2014). RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In C. Bizer, T. Heath, S. Auer, & T. Berners-Lee (Eds.), *Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, April 8, 2014.* Retrieved from http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutiérrez, C., … Zimmermann, A. (2020).

Knowledge Graphs. *CoRR, abs/2003.0*. Retrieved from https://arxiv.org/abs/2003.02320
Kellogg, G., Longley, D., & Champin, P.-A. (2020). *JSON-LD 1.1*.
Knublauch, H., & Kontokostas, D. (2017). Shapes Constraint Language (SHACL). Retrieved from W3C website: https://www.w3.org/TR/shacl/
Schreiber, G., & Raimond, Y. (2014). Rdf 1.1 Primer. Retrieved from W3C Working Group Note 24 June 2014 website: https://www.w3.org/TR/rdf-primer/
Seaborne, A., & Harris, S. (2013). SPARQL 1.1 Query Language. Retrieved from W3C Recommendation website: https://w2.syronex.com/jmr/w3c-biblio