Enhanced and Scalable RDF Validation Techniques for Dataspaces

Paul Moosmann[1,*], Johannes Theissen-Lipp[1,2] and Christoph Lange[1,2]

[1]Fraunhofer Institute for Applied Information Technology FIT, Sankt Augustin, Germany
[2]RWTH Aachen University, Aachen, Germany

Abstract

With the exponential growth of data on the Internet, the need to control data, especially after it has been shared, has become increasingly important. Dataspaces promise data sovereignty as a solution to this challenge, providing a decentralized environment where participants can share data while retaining control over its use. In such environments, where descriptive vocabularies and resource descriptions are decentralized and exchanged among participants, ensuring accurate and correct information becomes essential for integrated data use in mature software solutions. Validation principles play a crucial role in ensuring that the diverse information exchanged conforms to specified standards.

We enhance validation approaches to facilitate the common shape of information in decentralized environments such as dataspaces. We improve the validation techniques (1) through inference optimization and (2) introducing SHACL-SPARQL shortcut templates for easy expression of commonly used constraints. We evaluate the effectiveness of our approach through runtime measurements and application experiments with a Semantic Web expert. This approach enables accurate and scalable solutions, as shown by our evaluation, which demonstrates the practicality and scalability. Using state-of-the-art techniques such as knowledge inference, our method ensures data quality and usability while maintaining ease of use and inclusion for a wide range of dataspace participants, including non-experts. Our proposed two-step method improves validation techniques by optimizing RDF validation, leading to faster execution times of the validation process and reducing errors when implementing more complex validation shapes.

Keywords

Dataspaces, Scalable Data Validation, SHACL, RDF Inference

1. Introduction and Motivation

In the last years, the topic of dataspaces has become increasingly relevant, leading to multiple large-scale cross-domain dataspace initiatives being launched in Europe [1]. The most prominent examples include the International Data Spaces [2, 3] and Gaia-X [4, 5]. Concrete, domain-specific implementations of these initiatives include examples such as the Mobility Data Space [6] for the International Data Spaces or the automotive network Catena-X [7] for Gaia-X. Dataspaces offer approaches to exchange heterogeneous data from heterogeneous sources by supporting multiple data models and providing mechanisms to query and analyze the data, thereby discovering relationships amongst the data. However, an integrated use of data (e.g., in mature software solutions) demands precise and correct data. This requires validation principles to ensure that the heterogeneous information exchanged in dataspaces conforms to certain specifications. To achieve this, dataspaces use Semantic Web technologies such as SHACL [8]. The correct implementation of so-called *validation shapes*, which define the expected form of data, can be a non-trivial task for dataspace users who are not Semantic Web experts. This is especially true when validating against more complex constraints that are not natively supported by validation languages, and require the use of more advanced features, such as SHACL-SPARQL [8].

First International Workshop on Scaling Knowledge Graphs for Industry, co-located with 20th International Conference on Semantic Systems (SEMANTICS), Amsterdam, Sept. 17-19, 2024

*Corresponding author.
paul.moosmann@fit.fraunhofer.de (P. Moosmann); theissen-lipp@dbis.rwth-aachen.de (J. Theissen-Lipp); christoph.lange-bever@fit.fraunhofer.de (C. Lange)
0009-0005-2114-8578 (P. Moosmann); 0000-0002-2639-1949 (J. Theissen-Lipp); 0000-0001-9879-3827 (C. Lange)

In this paper, we address the requirements for an easy-to-use validation process and develop validation approaches to facilitate common information modeling in the decentralized environment of dataspaces. In this environment, two different sorts of data exist, that can be validated. There is (1) content data, meaning the actual payload data that is being exchanged and (2) metadata, which is data providing additional information about the data itself, e.g., who provided the data or what standards the data conforms to. The validation approach we suggest in this paper can be applied to both payload data and metadata, though we focused on metadata in our evaluation. We propose sound but convenient methods to facilitate validation in dataspaces. *Sound* in this context refers to precise and scalable solutions, including state-of-the-art solutions using advanced techniques such as knowledge inference to ensure the best possible results, while *convenient* refers to ease of use and inclusion of a wide range of users of a data space, including non-experts. We achieve this by simplifying the use of SHACL-SPARQL constraint shapes and by suggesting an inference step to pre-process the shapes. Therefore, this paper contributes:

- **Shortcuts for Commonly Used Constraints**: We provided shortcuts in the form of template-based properties for commonly used constraints that are not yet natively supported by validation languages, simplifying and unifying their application for users.
- **A Scaling Inference Solution for Validation**: Our modification of the current common practice of RDF validation with SHACL applies inference only once on the shape graph instead of on each instance.

This remainder of this paper first studies the state of the art in validation languages and techniques, and then presents our approach, results, and evaluation addressing the stated requirements.

2. Background and Related Work

The state of the art in the area of our work includes validation languages (cf. section 2.1) that allow users such as dataspace participants to design and evaluate constraints to ensure the conformance of resources in dataspaces. These resources, ranging from data to services to participant descriptions, can be validated against arbitrary specifications, including local agreements or global standards. The state of the art in validation solutions and techniques (cf. section 2.2) includes software applications as well as common practices for their application.

2.1. Validation Languages

The validation of RDF [9] data is an essential step of the linked data lifecycle to ensure high data quality. There are several languages available, aiming to enable extensive validation possibilities. To get an overview of the most relevant languages currently available for RDF validation, Dominik Tomaszuk conducted a survey "perform[ing] an overview and comparison of current options for RDF validation" in 2017 [10]. In this survey, he identified five different validation tools, ShEx [11], SHACL [8], ReSH [12], DSP [13], and SPIN [14], as the most important validation languages. He compared these languages, as well as the built-in validation capabilities of OWL [15], regarding their expressiveness of 17 commonly used constraints. Since SHACL was strongly influenced by SPIN and can be regarded as its legitimate successor [16], we will not evaluate both of these languages, but solely SHACL. Table 1 presents the results of our evaluation.

The metrics for evaluating the available RDF validation languages for our work are as follows. Expressiveness is ranked by how many of the 17 commonly used constraints [10] can be expressed, resulting in a score of "not very expressive" (–), "expressive" (+), and "very expressive" (++). In addition, we compare the validation languages in terms of their standardization, use in existing dataspace initiatives, and existing inference support. These are simple yes/no measures.

SHACL is a constraint language for RDF. It can express 16 out of 17 commonly used constraints. It is a W3C recommendation and is applied in dataspace initiatives such as Gaia-X and the International Data

Spaces. Existing SHACL validators offer support for RDFS inference. *ShEx* is a language for describing RDF graph structures that can prescribe conditions that RDF data graphs must meet to be considered *conformant*. It can express 16 out of 17 commonly used constrains. Any inference must be done on the RDF graph separately, the ShEx processor itself does not interact with any inference mechanism [17]. *ReSH* is a high-level RDF vocabulary for specifying the shape of RDF resources. It is only able to express 9 out of 17 commonly used constrains. There is no additional inference support, besides the general possibility to perform inference on the RDF graph directly. *DSP* is used to define structural constraints on data. It is only able to express 10 out of 17 commonly used constraints. There is no additional inference support, besides the general possibility to perform inference on the RDF graph directly. *OWL* is a language to describe RDF graph structures used in dataspace initiatives such as Gaia-X and the International Data Spaces. It is part of the W3C's Semantic Web technology stack. While it has built-in validation capabilities, it can only express 13 out of 17 commonly used constraints. It is possible to apply inference mechanisms on graphs described using OWL.

Table 1
Requirements and metrics for validation languages. *Expressiveness* in this context refers to the number of expressible constraints from the list of 17 commonly used constraints.

Validation Language	Expressiveness	Standardization	Dataspace Use	Inference Support
SHACL [8]	++	✓	✓	✓
ShEx [11]	++	✗	✗	✗
ReSH [12]	–	✗	✗	✗
DSP [13]	–	✗	✗	✗
OWL [15]	+	✓	✓	✓

Based on the results shown in Table 1, we use SHACL in this work because it best meets our requirements.

2.2. Solutions and Techniques for Validation

Since we have identified SHACL as the most appropriate validation language for our requirements, we will focus in this section on solutions that implement SHACL. Apache Jena SHACL [18] implements both SHACL Core and SHACL SPARQL constraints and provides a reader and writer for a compact SHACL syntax. The TopBraid SHACL API [19] is an open-source implementation of SHACL based on Apache Jena, which performs SHACL constraint checking and rule inferencing. There also exists a data governance tool using this API, the TopBraid EDG [20]. A GitHub issue opened in February 2021 [21] shows that the topic of inference for RDF validation can be further improved. The issue raises the question of subclass inference by SHACL. The answers suggest that today the only way to perform this kind of inference is to import the whole ontology (or ontologies) into the data graph, which does not always seem to be a high-performance solution. Additionally, there is no easily accessible documentation available that shows what kind of inference is done.

There are web-based validators available for simple validation tasks. E.g., the SHACL Playground [22] is implemented by TopQuadrant, who are also responsible for the TopBraid EDG mentioned above. There exists an updated implementation of the SHACL Playground that can be found under the name of Zazuko SHACL Playground [23]. Another web-based validator is provided as part of the European Commission's DG DIGIT Interoperability Test Bed [24]. The interface offers a simple possibility to upload a shape and data graph to conduct the validation. Apart from the data validator, they also offer a shape validator [25], which checks if the provided SHACL shape confirms to certain rules, namely the core W3C syntax rules, the extended W3C syntax rules, and the extended W3C syntax rules and best practices. These web-based validators have in common that they were developed for simple validation tasks and do not support more advanced tasks. For example, it is not possible to import ontologies into the data graph for inference or use advanced features such as SHACL-SPARQL for the SHACL Playground.

3. Approach

Our approach to enhancing validation techniques for dataspaces consists of two steps: The introduction of shortcut templates (cf. section 3.1) and the optimization by inference (cf. section 3.2). Our approach also suggests a combination of these two steps into one solution (cf. section 3.3).

3.1. Introducing Shortcut Templates for Constraints

We begin by addressing the challenge of dealing with commonly used constraints that are not natively supported by validation languages (see figure 1). The current state-of-the-art approach for SHACL is to manually formulate SHACL-SPARQL constraints, which can be complex and ambiguous (since one constraint can be modelled using varying SHACL-SPARQL constructs). To simplify and unify this process, we propose the introduction of shortcuts in the form of new template-based properties. Our proposed methodology is shown in figure 2. The approach involves (1) analyzing the feasibility of common yet unsupported constraints, which were presented by Hartmann in [26], (2) designing templates for them, (3) aggregating them for easy use, and (4) implementing a demonstrator for them.

```
shapes:ResourceShape
shapes:ResourceShape
shapes:ResourceShape
  a sh:NodeShape ;
  sh:targetClass ids:Resource ;
  sh:constraint [
  sh:select """
      SELECT $this
      WHERE {
        FILTER NOT EXISTS {
          $this rdfs:label ?value .
          FILTER (isLiteral(?value) &&
                langMatches(lang(?value), 'en'))
        }
      }
      """ ; ] .
```

(a) Common approach (baseline)

```
shapes:ResourceShape
  a sh:NodeShape ;
  sh:targetClass ids:Resource ;
  sst:languageTagDefinition "rdfs:label, en" .
```

(b) Our approach

Figure 1: The first part of our two-part approach is to introduce shortcuts for commonly used constraints that are not yet natively supported by validation languages. For each non-standard constraint, the current state of the art for SHACL (a) is that users must formulate SHACL-SPARQL constraints, which are often complex and ambiguous. We introduce shortcuts (b) in the form of new template-based properties to simplify and unify this process.

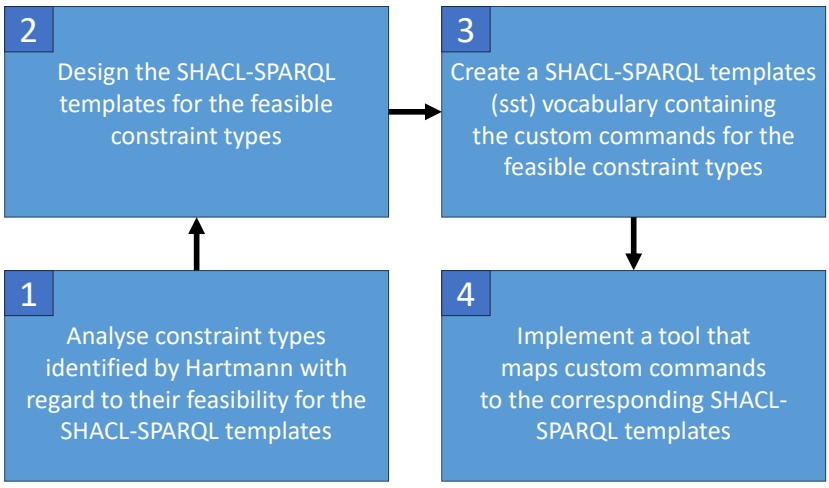
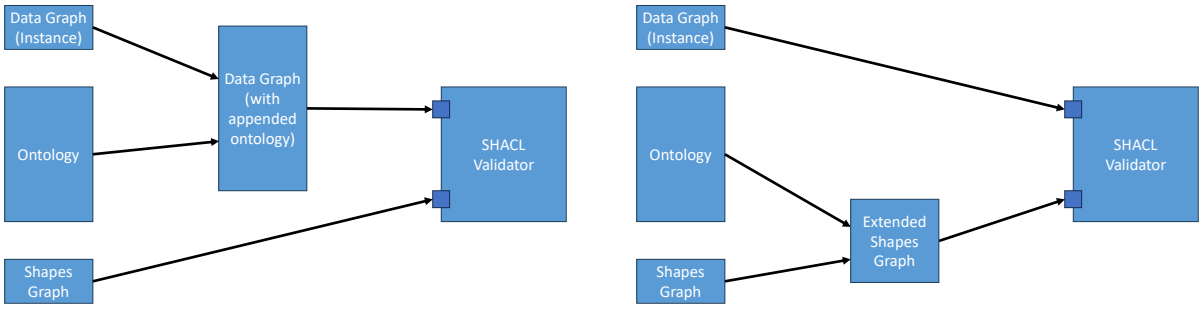


Figure 2: The methodology of the second part of our two-part approach, which introduces SHACL-SPARQL templates as shortcuts for commonly used constraints not natively supported by SHACL. We (1) analyze the feasibility of common but unsupported constraints [26], (2) design templates for them, (3) aggregate them as shortcuts for easy use, and (4) implement a demonstrator that supports them.

These shortcut templates serve as intuitive bridges between unsupported constraints and native SHACL validators, significantly reducing the cognitive overhead for users. By encapsulating complex constraints into reusable templates, we enable users to express validation rules more concisely and effectively. In addition, the adoption of standardized templates improves interoperability and promotes best practices across different validation scenarios.

3.2. Validation Enhancement through Inference Optimization

In the second part of our approach, we address the inefficiencies inherent in current RDF validation processes (see figure 3). Applying inference as part of the RDF validation with SHACL usually involves appending the entire ontology file needed for inference to each data instance. This significantly increases the size of each instance to be validated. Our proposed solution modifies this practice by applying the inference only once to the shapes graph. This optimization results in a shapes graph of slightly higher complexity but eliminates the need for expensive inference on each data graph during validation. With this approach, we aim for a more scalable solution for recurrent validation against the same shapes graph.



(a) Validation without prior inference (baseline)

(b) Validation with prior inference (our approach)

Figure 3: The second part of our two-part approach is to modify a current common practice for RDF validation with SHACL. In the baseline (a), the entire ontology file needed for inference is appended to each instance and fed to a validator, increasing the size of each instance to be validated enormously. We propose an approach (b) that instead applies inference only once to the shapes graph.

Our approach not only reduces the computational overhead but also increases the agility of the validation process. By decoupling the inference step from individual data instances, we aim to create a more streamlined and resource-efficient workflow. This optimization could prove particularly advantageous for large-scale dataspaces, since they require frequently performed validation tasks.

3.3. Combining the Solutions

Combining these two solutions into one is the final step of our approach (see figure 4). The input data is pre-processed by our proposed scaling inference solution, which optimizes the shapes graph for subsequent validation. Subsequently, our constraint template mapping solution outputs native SHACL shape graphs using SHACL-SPARQL. This ensures compatibility with any native SHACL validator supporting SHACL-SPARQL, therefore facilitating seamless validation of data against predefined constraints.

4. Implementation

In this section, we detail the implementation of our previously described approach. Our work enhances the validation process of existing validation tools, such as the command-line tool pySHACL [27] by implementing easy-to-use commands for complex constraints and performing an inference step pre-validation. Since this is a two-step approach, we present our results for each step separately.

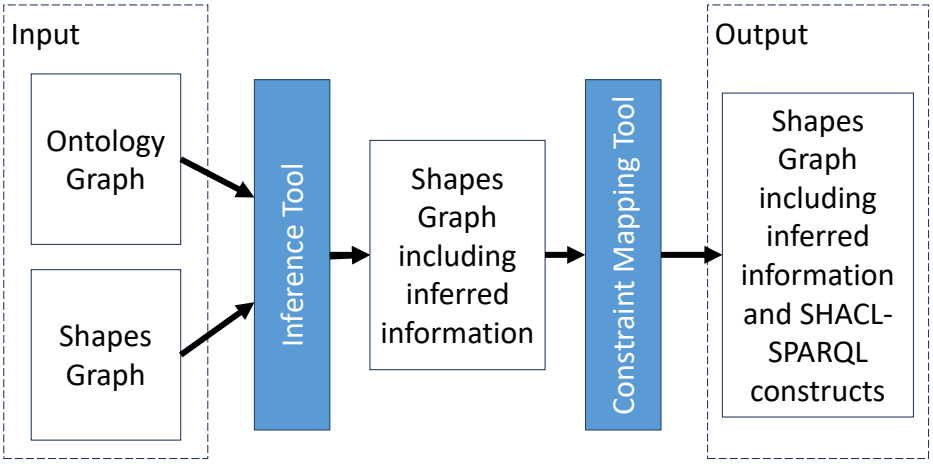


Figure 4: The complete solution of our two-step approach. For input data to be validated, our proposed scaling inference solution (see figure 3) first manipulates the shapes graph, before our proposed constraint template mapping solution (see figure 1) makes it compatible with any native SHACL validator, so that the actual validation can be performed seamlessly.

4.1. Resulting Shortcut Templates for Constraints

The results of this work include 16 constraint templates implementing commonly used constraints [10] using SHACL-SPARQL. This way, we can utilize the built-in SPARQL support of the SHACL validation language. Using these templates, a potential user wanting to validate a data instance does not need to implement a complex SHACL-SPARQL solution. Instead, the user only needs to use a simple command that is replaced by the corresponding SHACL-SPARQL template during the mapping process. Listing 1 shows one of the created constraint templates. It implements a constraint on language tags for a data property.

```
1  :CountryShape a sh:Shape ;
2  sh:scopeClass :Country ;
3  sh:constraint [
4  sh:message "Values of 'germanLabel' must hava a German language tag." ;
5  sh:sparql """
6  SELECT $this ($this AS ?subject) (:germanLabel AS ?predicate)
7         (?value AS ?object)
8  WHERE { $this :germanLabel ?value .
9        FILTER(isLiteral(?value) ||
10              !langMatches(lang(?value), "de"))}""";].
11
12 :ValidCountry a :Country ; :germanLabel "Deutschland"@de .
13 :InvalidCountry a :Country ; :germanLabel "Germany"@en .
```

Listing 1: Example SHACL-SPARQL template to validate that a certain language tag (German) is present for a data property. A valid example instance (line 12) has a German language tag (@de) and an invalid instance (line 13) has only an English language tag (@en).

Figure 5a shows an example of a possible input graph for the constraint mapping tool. This SHACL shapes graph defines a constraint on the cardinality of English language tags on the `rdfs:label` property by using one of the custom commands defined by this work. The constraint mapping tool then maps this command onto the corresponding SHACL-SPARQL construct as defined in the constraint template designed by us. Figure 5b shows the result of this mapping process. To achieve the mapping from Figure 5a to Figure 5b, a simple command-line command with two input parameters is sufficient.

The first is the path to the SHACL shapes graph containing our custom `sst` commands. The second is the output location for the created SHACL shapes graph containing the SHACL-SPARQL templates. A possible command line execution is:

```
python Mapping.py pathTo/input.shacl.ttl pathTo/output.shacl.ttl
```

This mapping is implemented using a graph-based approach. The graph is searched for triples using one of our custom commands as an edge between two nodes. When finding such a triple, it is replaced by a new triple using the `sh:sparql` command as predicate and the corresponding SHACL-SPARQL template as object. Hereby, the input file is imported as a graph data structure using the Python package RDFlib[1].

```
@prefix gax: <https://registry.lab.gaia-x.eu/development/api/trusted-shape-registry/v1/shapes/
    jsonld/trustframework#> .
@prefix sh: <http://www.w3.org/ns/shacl> .
@prefix sst: <http://example.org/sst-vocab> .

sh:ProviderShape a sh:NodeShape ;
sst:languageTagCardinalityMin "rdfs:label,rdfs:comment,en,1";
sh:targetClass gax:Provider .
```

(a) Input

```
@prefix gax: <https://registry.lab.gaia-x.eu/development/api/trusted-shape-registry/v1/shapes/
    jsonld/trustframework#> .
@prefix sh: <http://www.w3.org/ns/shacl> .

sh:ProviderShape a sh:NodeShape ;
sh:sparql [ sh:message "Values of the constraint 'language tag cardinality min' [...]" ;
sh:select """
SELECT $this
WHERE {
  SELECT (COUNT(?value) as ?count)
  WHERE {
    { $this rdfs:label ?value } UNION { $this rdfs:comment ?value }
    FILTER(isLiteral(?value) && langMatches(lang(?value), 'en'))
  }
  GROUP BY $this
}
HAVING (SUM(?count) < 1)
""" ] ;
sh:targetClass gax:Provider .
```

(b) Output

Figure 5: Example execution for a constraint that all labels and comments have at least one English value, using our proposed constraint mapping tool. The tool transforms an introduced shortcut (see line 6 in (a)) into corresponding SHACL-SPARQL expressions (see lines 5–17 in (b)), which can be used natively in any SHACL validator.

[1] https://rdflib.readthedocs.io/en/stable/

4.2. Resulting Inference Optimization

Our solution for performing the inference as a preprocessing step is implemented as a tool similar to the mapping tool above. Again, use a graph-based approach consisting of two tasks. The first task is searching the ontology graph for all the sub-class and sub-property relations. The second task is searching the shapes graph for occurrences of parent classes and properties and adding further connections to cover all related sub-classes and sub-properties. For the second task, the tool identifies specific connection types in the shape graph. Figure 6 shows a minimal example of what the input and output graphs of the inference tool look like when performing a simple sub-class inference of the `sh:targetClass`. Analogous to the realization of the constraint templates, our transitive inference tool is realized by a command-line tool implemented in Python 3. More detailed, we use the Python library RDFlib, which enables an implementation of our graph-based approach. To perform the inference step, a simple command-line command with three input parameters is sufficient. The first is the path to the ontology graph containing additional information, the second is the path to the SHACL shapes graph, and the third is the output location for the extended SHACL shapes graph. A possible command line execution is:

```
python mapping.py ontology.ttl shape.ttl newShape.ttl
```

```
1  @prefix ex: <http://example.org/> .
2  @prefix sh: <http://www.w3.org/ns/shacl#> .
3
4  ex:PersonShape
5      a sh:NodeShape ;
6      sh:targetClass ex:Person ;
7      sh:property [
8          sh:path ex:hasName ;
9          sh:minCount 1 ;
10     ] .
```

(a) Input

```
1  @prefix ex: <http://example.org/> .
2  @prefix sh: <http://www.w3.org/ns/shacl#> .
3
4  ex:PersonShape
5      a sh:NodeShape ;
6      sh:targetClass ex:Person, ex:Student ;
7      sh:property [
8          sh:path ex:hasName ;
9          sh:minCount 1 ;
10     ] .
```

(b) Output

Figure 6: Example execution of an sub-class inference. Our tool updates the SHACL graph using the information that `ex:Student` and is a sub-class of `ex:Person`. This is implemented by adding `ex:Student` as `sh:targetClass` (cf. line 6 in (b)).

5. Evaluation

To assess the effectiveness and scalability of our proposed enhanced validation techniques, we conducted an evaluation using example instances from the Gaia-X framework for dataspaces. We decided to do this, to show the feasibility of our approach in real-world application scenarios. These instances, as

shown in figure 7, represent typical data providers within the Gaia-X ecosystem, showing properties with data attributes and links to related objects. Understanding the details of these instances is relevant for evaluating the performance of our validation techniques. Note that the names and data of these providers have been anonymized for this paper.

Properties \ Instances	Provider 1	Provider 2	Provider 3	Provider 4
ID	gax:Provider1	gax:Provider2	gax:Provider3	gax:Provider4
rdfs:type	gax:Provider	gax:Provider	gax:Provider	gax:Provider
gax:hasName	Provider 1	Provider 2	Provider 3	Provider 4
gax:hasSalesTaxID	101	102	103	104
ex:hasProjectPartner	gax:Provider2	gax:Provider1		
ex:trusts	gax:Provider1	gax:Provider2	gax:Provider3	gax:Provider4
ex:isSubsidiaryOf			gax:Provider1	gax:Provider1, gax:Provider3
rdfs:label	Provider 1	Provider 2	Provider 3	Provider 4

Figure 7: Overview of the four example instances of providers from the Gaia-X dataspace used in this evaluation. Each provider instance contains properties defining the type, name, sales tax ID, and label. Further properties describe the relation amongst the different providers, such as subsidiary or project partner relations, and if a provider trusts the other providers.

5.1. Evaluation of the Impact of Shortcut Templates for Constraints

To evaluate the impact of the constraint template enhancement on the time required for the validation process and the error proneness of this process, we conducted a hands-on experiment that simulated a real-world work situation. For this purpose, we asked a Semantic Web expert to implement a defined set of constraints in SHACL once without the constraint mapping tool and once with the constraint mapping tool. The developer's skills included working with SHACL on a weekly basis and having basic knowledge of SPARQL.

We structured the experiment in the following way. Given was a data graph describing the four different provider instances shown in figure 7. The data graph was supposed to satisfy the following four constraints: (C1) each provider trusts itself, (C2) the legally binding name of a provider must be unique, (C3) the property `ex:hasProjectPartner` is symmetric, and (C4) at least one label with an English language tag must be defined. An example data graph intentionally contained one error for each of these constraints, meaning a validation with a correct shapes graph should lead to four validation errors. Given the data graph and the constraint requirements, the developer was tasked with creating a SHACL shapes graph without using the constraint mapping tool. After 30 minutes, a validation of the data graph was conducted. For the second part of the experiment, the developer used the constraint mapping tool in the form of a Python command line tool. In addition, we provided written documentation on how to use the tool. Again, the allotted time was 30 minutes. The results of the experiment are summarized in Table 2 and show that our approach led to improvements in both the time needed to build the shapes graph and the correctness of the validation process.

5.2. Evaluation of the Inference Optimization

In the second part of our evaluation, the inference optimization, we focused on measuring the execution times of our proposed validation techniques under different scenarios. Table 3 summarizes the results of several runs of our experiment, comparing the baseline common approach with our proposed method. Notably, our approach achieves a significant reduction in execution times for individual data graphs, demonstrating its efficiency and scalability. However, the execution times for a merged data graph remain relatively unchanged between the baseline and our approach. Figure 8 visualizes the average

	Without the constraint mapping tool	With the constraint mapping tool
Implementation of C1	incorrect implementation	successfully implemented
Implementation of C2	not implemented	successfully implemented
Implementation of C3	not implemented	successfully implemented
Implementation of C4	incorrect implementation	successfully implemented
Time needed	Experiment stopped after 30 minutes	Done after 15 minutes

Table 2
Overview of the experiment results, determining the utility of our constraint mapping tool.

execution times of all runs for a comprehensive analysis. Table 3 shows a graphic representation of the average values of our different experiment executions. These results validate the practicality and scalability of our proposed validation approaches for application in real-world scenarios.

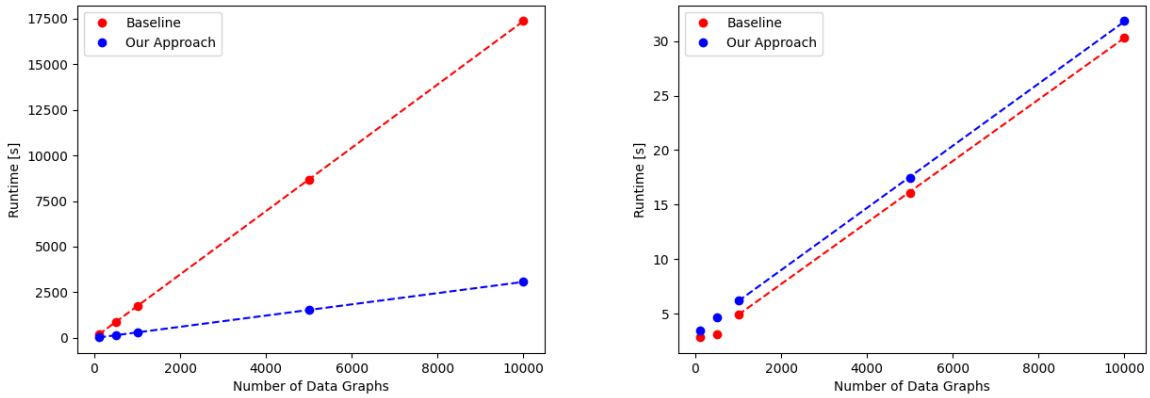
Table 3
Execution times measured in our experiments. In both scenarios, *individual data graphs* and *merged data graph*, we compared the state-of-the-art common approach as a baseline with our approach. Scalability is measured by a growing number of 100–10000 input data graph files. Each cell on the right contains three measurements representing repeated experiments. While our approach does not change the execution times for a merged data graph, it significantly reduces the execution times for individual data graphs to about 1/5.

		Number of Files				
		100	**500**	**1000**	**5000**	**10000**
individual data graphs	Baseline	171.85	882.09	1737.95	8600	17200
		177.36	884.06	1808.80	8850	17700
		172.00	866.37	1658.90	8600	17200
	Our Approach	31.54	142.45	289.93	1600	3200
		30.41	144.34	298.70	1500	3000
		30.2	145.57	258.76	1500	3000
merged data graph	Baseline	2.82	3.02	5.12	16.24	30.68
		2.76	3.17	4.73	15.83	30.21
		2.84	3.19	5.02	16.08	30.15
	Our Approach	3.34	4.47	5.99	17.26	31.48
		3.44	4.67	6.09	18.73	32.08
		3.55	4.82	6.48	17.41	31.96

6. Conclusion and Future Work

In conclusion, we successfully addressed the rising data validation requirements by proposing a scaling inference solution and a constraint template mapping solution. This facilitates common information design in decentralized environments such as dataspaces by making validation techniques more scalable and usable. An evaluation in a real-world scenario with anonymized participants from the Gaia-X dataspaces demonstrated feasibility, scalability, and practical impact. The contributions of this paper include:

- **A Scaling Solution for Validation**: Our modification of the current common practice of RDF validation with SHACL applies inference only once on the shapes graph instead of on each instance. The shapes graph becomes slightly more complex, but expensive inference on each data graph is avoided. This contributes to the requirements by making recurrent validation against the same shapes graph more scalable.



(a) Validating individual data graphs

(b) Validating merged data graph

Figure 8: Visualization of the average execution times of our experiments (see table 3). While our approach does not change the execution times for a merged data graph (b), it significantly reduces the execution times for individual data graphs to about 1/5 and scales linearly.

- **Shortcuts for Commonly Used Constraints**: We provided shortcuts in the form of template-based properties for commonly used constraints that are not yet natively supported by validation languages, simplifying and unifying their application for users. We have introduced SHACL-SPARQL templates, which are used by our proposed mapping tool to enable native support by any SHACL validator. This contributes to the requirements by allowing a wide range of dataspace users, including non-experts, to formulate even complex constraints for validation.
- **Combined Approach**: By combining inference optimization with the introduction of constraint templates, our two-step approach provides a holistic solution for improving dataspace validation techniques. We not only address efficiency concerns but also prioritize usability and interoperability, thereby advancing the state of the art in RDF validation methodologies. This approach lays the foundation for scalable and robust validation processes in the context of evolving dataspaces and Semantic Web applications.

The contributions provide reliable guarantees for an integrated use data or services in applications on the domain layer of dataspaces, such as APIs. The proposed scaling but convenient solutions for validation in dataspaces allow the diverse participants of dataspaces, including non-experts, to use shortcuts for expressing common validation constraints and to validate these much faster with standard validators. The proposed enhanced validation techniques improve interoperability by providing guarantees for all kinds of interfaces between participants or services in dataspaces, which contributes to a better common understanding in dataspaces.

Future work can expand the pool of constraint templates with corresponding templates for property-based constraints. Concerning the inference approach introduced by this work, our tool currently only applies transitive inference for the properties `rdfs:subClassOf` and `rdfs:subPropertyOf`. Future work can extend the considered targets of the inference process to achieve even better validation results. It can also include developing a user interface for our command-line tools to make them even more accessible.

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612. Funded by the FAIR Data Spaces project of the German Federal Ministry of Education and Research (BMBF) under the grant number FAIRDS05. We thank Jens Lehmann for supervising the master thesis underlying this research.

References

[1] J. Theissen-Lipp, M. Kocher, C. Lange, S. Decker, A. Paulus, A. Pomp, E. Curry, Semantics in dataspaces: Origin and future directions, in: Companion Proceedings of the ACM Web Conference 2023, WWW '23 Companion, Association for Computing Machinery, New York, NY, USA, 2023, p. 1504–1507. URL: https://doi.org/10.1145/3543873.3587689. doi:10.1145/3543873.3587689.

[2] International Data Spaces e. V., International Data Spaces, https://www.internationaldataspaces.org/, 2016. [Accessed 29.04.2024].

[3] S. Bader, J. Pullmann, C. Mader, S. Tramp, C. Quix, A. Müller, H. Akyürek, M. Böckmann, B. Imbusch, J. Theissen-Lipp, S. Geisler, C. Lange, The International Data Spaces Information Model - An Ontology for Sovereign Exchange of Digital Content, in: Proceedings of the 19th International Semantic Web Conference, Springer International Publishing, Cham, 2020, pp. 176–192. URL: https://doi.org/10.1007/978-3-030-62466-8_12. doi:10.1007/978-3-030-62466-8_12.

[4] Gaia-X European Association for Data and Cloud AISBL, Gaia-X: Vision & Mission, https://gaia-x.eu/what-is-gaia-x/vision-and-mission/, 2020. [Accessed 08.08.2023].

[5] Plattform Industrie 4.0, Project GAIA-X: A Federated Data Infrastructure as the Cradle of a Vibrant European Ecosystem, Technical Report, Federal Ministry for Economic Affairs and Energy (BMWi), D-11019 Berlin, Germany, 2019.

[6] DRM Datenraum Mobilität GmbH, Mobility Data Space, https://mobility-dataspace.eu/, 2021. [Accessed 09.05.2024].

[7] Catena-X Automotive Network e.V., Catena-X Automotive Network, https://catena-x.net/en/, 2021. [Accessed 09.05.2024].

[8] H. Knublauch, D. Kontokostas, Shapes constraint language (SHACL), W3C Recommendation, W3C Working Group, 2017. Retrieved from https://www.w3.org/TR/shacl/, on 23.04.2024.

[9] G. Schreiber, Y. Raimond, F. Manola, E. Miller, B. McBride, RDF 1.1 Primer, W3C Recommendation, W3C Working Group, 2014. Retrieved from https://www.w3.org/TR/rdf11-primer/, on 23.04.2024.

[10] D. Tomaszuk, Rdf validation: a brief survey, in: International Conference: Beyond Databases, Architectures and Structures, Springer, 2017, pp. 344–355.

[11] E. Prud'hommeaux, J. E. Labra Gayo, H. Solbrig, Shape expressions: an RDF validation and transformation language, in: SEMANTICS, 2014, pp. 32–40. doi:10.1145/2660517.2660523.

[12] A. Ryman, Resource Shape 2.0, W3C Member Submission, W3C Working Group, 2014. Retrieved from https://www.w3.org/submissions/shapes/, on 23.04.2024.

[13] M. Nilsson, Description Set Profiles: A constraint language for Dublin Core™ Application Profiles, Technical Report, DCMI, 2008. Retrieved from https://www.dublincore.org/specifications/dublin-core/dc-dsp/, on 23.04.2024.

[14] H. Knublauch, J. A. Hendler, K. Idehen, SPARQL Inferencing Notation, W3C Member Submission, W3C Working Group, 2011. Retrieved from https://www.w3.org/submissions/2011/SUBM-spin-overview-20110222/, on 23.04.2024.

[15] W3C OWL Working Group, OWL 2 Web Ontology Language, W3C Recommendation, W3C Working Group, 2011. Retrieved from https://www.w3.org/TR/2012/REC-owl2-overview-20121211/, on 23.04.2024.

[16] H. Knublauch, Spin webpage, https://spinrdf.org/, 2011. Accessed: 2024-04-23.

[17] J. E. L. Gayo, E. Prud'hommeaux, I. Boneva, D. Kontokostas, Validating RDF Data - Chapter 7: Comparing ShEx and SHACL, Springer International Publishing, 2018. doi:10.1007/978-3-031-79478-0.

[18] The Apache Software Foundation, Apache jena shacl, https://jena.apache.org/documentation/shacl/index.html, 2011. Accessed: 2024-04-23.

[19] TopQuadrant, Topbraid shacl api, https://github.com/TopQuadrant/shacl/tree/d566c6b955cb8ec63ca32129bfb41b358ac07e31, 2016. Accessed: 2024-04-23.

[20] TopBraid, Topbraid enterprise data geovernance, https://www.topquadrant.com/products/topbraid-enterprise-data-governance/, 2017. Accessed: 2024-04-23.

[21] R. Palma, Sub-class inference using shacl, https://github.com/TopQuadrant/shacl/issues/108, 2021.

Accessed: 2024-04-23.
[22] H. Knublauch, Shacl playgound, https://shacl.org/playground/, 2017. Accessed: 2024-04-23.
[23] Zazuko GmbH, Zazuko shacl playgound, https://shacl-playground.zazuko.com/, 2021. Accessed: 2024-04-23.
[24] European Commission's DG DIGIT, Dg digit shacl validator, https://www.itb.ec.europa.eu/shacl/any/upload, 2020. Accessed: 2024-04-23.
[25] European Commission's DG DIGIT, Dg digit shacl shape validator, https://www.itb.ec.europa.eu/shacl/shacl/upload, 2020. Accessed: 2024-04-23.
[26] T. Hartmann, Validation framework for rdf-based constraint languages, Ph.D. thesis, Karlsruhe Institute of Technology, Germany, 2016.
[27] A. Sommer, N. Car, A python validator for shacl, https://github.com/RDFLib/pySHACL, 2018. doi:10.5281/zenodo.10958008, accessed: 2024-04-23.

A. Data availability statement

The tools developed as part of this work can be accessed via https://github.com/moosmannp/Enhancing-SHACL-Validation-Through-Constraint-Templates-and-Inference.