**SHAPEness**: a SHACL-driven RDF Graph Editor

Rossana Paciello\textsuperscript{a,b,}\textsuperscript{*}, Daniele Bailo\textsuperscript{a,c}, Luca Trani\textsuperscript{b}, Valerio Vinciarelli\textsuperscript{a}, Manuela Sbarra\textsuperscript{c}, Lorenzo Fenoglio\textsuperscript{c}, Sara Capotosti\textsuperscript{c}

\textsuperscript{a} European Plate Observing System, EPOS-ERIC, Rome, Italy  
\textsuperscript{b} Department of R&D Seismology and Acoustics, Royal Netherlands Meteorological Institute (KNMI), Utrechtseweg 297, 3731 GA, De Bilt, The Netherlands  
\textsuperscript{c} Istituto Nazionale di Geofisica e Vulcanologia (INGV), Rome, Italy

E-mails: rossana.paciello@ingv.it, daniele.bailo@ingv.it, luca.trani@knmi.nl, valerio.vinciarelli@epos-eric.eu, manuela.sbarra@ingv.it, lorenzo.fenoglio@ingv.it, sara.capotosti@ingv.it

**Abstract.** The Shapes Constraint Language (SHACL) has been recently introduced as a W3C recommendation to define constraints (called *shapes*) for validating RDF graphs. In this paper a novel SHACL-driven multi-view editor is presented: SHAPEness. It empowers users by offering them a rich interface for assessing and improving the quality of RDF graphs. SHAPEness has been developed and tested in the framework of the European Plate Observing System (EPOS). In this context, the SHAPEness features have proven to be a valuable solution to easily create and maintain valid graphs according to the EPOS data model. The SHACL-driven approach underpinning SHAPEness, makes this tool suitable for a broad range of domains, or use cases, which structure their knowledge by means of SHACL constraints.

Keywords: SHACL shapes, RDF validation, knowledge graph quality, graphs visualization, metadata editor.

1. Introduction

Over the past decade, the use of standard semantic technologies and linked data principles have become general practice in structuring metadata and related data. For instance, semantic representations such as ontologies are applied to address FAIR data principles [22, 10], that require rich metadata descriptions for scientific datasets. These principles have gained consensus within scientific communities, fostered by pan-European initiatives such as the European Open Science Cloud (EOSC) [16], where they are used as driving concepts to support data interoperability among standardized repositories compliant to a shared set of requirements [14]. According to the FAIR principles, metadata can facilitate data exchange, data findability [14] and data access [5].

Resource Description Framework\textsuperscript{1} (RDF) is the most popular format for exchanging semantic (metadata, structured as dynamic and schemaless graphs. However, the RDF flexibility turns out to be the source of many data quality and knowledge representation issues. Furthermore, since the use of schema, ontology and constraint languages is not mandatory, there is often room for misunderstandings related to the structure of the data [15].

The validation of the structure of RDF graphs against a set of constraints is a requirement shared by many users with different expertise levels, and represents an actual and relevant research topic [12, 4].

\textsuperscript{*} Corresponding author. Email: rossana.paciello@ingv.it

\textsuperscript{1} https://www.w3.org/RDF/
Constraint languages, such as SHACL ² or ShEx³, meet this need by providing models and vocabularies for expressing structural and semantic relationships [11] — they typically support different types and multiple levels of severity.

The Shapes Constraint Language (SHACL) is a recent W3C recommendation and a powerful formalism to define a set of constraints (called shapes) on the content of RDF data graphs (called nodes) thereby improving their quality. SHACL addresses various use cases and application scenarios, for instance, it can be adopted to validate RDF data structures, as a template to model RDF and as an RDF structure query mechanism.

The modelling of such constraints is usually the realm of experts with a deep understanding of the target domain concepts (e.g. environmental science) who are also knowledgeable in the language’s syntax. In this way they are able to describe the desired data structure whilst considering the domain requirements.

Due to the potential size of the data structure and the complexity of the restrictions needed to model a knowledge domain, several approaches and tools have been proposed for supporting experts in the above-mentioned modelling task [9, 3, 7, 8].

Nevertheless, defining SHACL constraints is just the first step towards assessment and improvement of knowledge graphs quality [18]. It is equally important to provide visual tools to create, edit and maintain graphs compliant to such constraints, even when users have a limited knowledge of the domain, of its main concepts and data structures.

In this paper we present SHAPEness, a SHACL-driven desktop application that provides users with a rich interactive and user-friendly environment where they can generate, visualize and work with RDF data and structures complying to defined SHACL constraints (shapes).

RDF data graphs can be easily created by loading a set of shapes into the SHAPEness application; graph properties can be edited and visualized using simple forms; and the resulting graphs can be inspected and serialized to RDF/Turtle⁴ format.

SHAPEness’ features are provided by combining three different types of views in a single user interface: graph-based, form-based, tree-based. This approach enables users to easily perform their tasks by hiding data structure complexity, and thus overcoming the lack of technical expertise.

The SHACL-driven mechanism, implemented by SHAPEness, aims at providing an innovative visual tool in creating and validating metadata as RDF graphs, suitable for any knowledge domain described by means of SHACL constraints. In this paper we prove its added value in the context of solid-Earth sciences.

The remainder of this paper is organized as follows: section 2 provides an overview of the existing tools for creating and validating RDF data, against a set of constraints. Section 3 describes the main features of SHAPEness. Section 4 gives a complete overview of the SHAPEness architecture. Section 5 describes the application of SHAPEness in a real case scenario. Section 6 is devoted to discussing the approach and results of user testing. Finally, Section 7 summarizes the paper and provides the directions for future developments.

2. Related tools

At present a limited set of visual tools is available for supporting users in the creation and validation of RDF data by means of user-friendly user interfaces. This might be due to the recent standardization of constraint languages for Linked Data such as SHACL.

This section introduces some of those tools and provides considerations about their supported features.

Schimatos [23] is a form-based web application that helps users to create and edit RDF data, validated against SHACL constraints. The web forms are automatically generated by using the SHACL content and the acquired data are stored in a SPARQL database. It offers also a client-side validation which allows to minimize the input of erroneous data. Schimatos serializes the created RDF data to RDF/Turtle format, but only for storing data in a SPARQL database. The software is available to download as an HTML+CSS/JavaScript package.

ActiveRaUL [6] is a web forms generator that helps users, with little or no knowledge about the Semantic Web, to create and maintain RDF data. It operates on a model defined according to the RDFa User Interface Language⁵ that consists of two parts: 1) a form model for describing the structure of a web form with different types of forms controls their associated operations; and 2) a data model for defining the structure of the exchanged data as RDF statements which are referenced from the form model via

---

² https://www.w3.org/TR/shacl/
³ https://github.com/shexSpec/shex/wiki/ShEx
⁴ https://www.w3.org/TR/turtle/
⁵ http://purl.org/NET/raul
a data binding mechanism. ActiveRaUL is able to create input forms directly using an arbitrary RDF ontology or an RaUL RDF model in order to acquire data from users and store them in a SPARQL database. However, ActiveRaUL does not provide users with a serialization of RDF data and does not support any types of validation of the edited metadata.

RDForms is a collection of javascript libraries for supporting users in the creation of a form-based RDF editor in a web environment. It is the sixth version of a project started in 2001 previously known as SHAME. Developers can use one or a combination of such libraries to build up a web application which includes: the editor, the validator, the linked data browser and the template editor. The template editor helps domain experts to define RDF templates by using a graphic interface. It also provides a converter to generate templates from RDF schemes or Description Set Profile (DSP). The editor generates web forms according to the defined templates and enables users to fill in the RDF data. The serialization format is RDF/XML or RDF/JSON and the RDF statements are shown in a dedicated area of the editor. RDForms offers a simple table visualization of the Linked Data resources. It supports a basic validation mechanism in order to generate a report with errors and warnings about the cardinality restrictions, as they are defined by RDForms-templates.

The VitroLib Metadata Editor is one of the outputs of the LD4L project (Liked Data 4 Libraries), a collaboration of Cornell, Harvard, Iowa, and Stanford Universities to promote the use and value of linked data in libraries. VitroLib extends Vitro that is an open-source community driven web application development platform best known as the software underlying the VIVO research networking tool. Vitro enables developers to build up a complete web platform to manage RDF resources. By leveraging the VitroLib extension, users can configure the platform, at installation time, in order to work with a specific SHACL file. VitroLib assists users in data editing and, since the data storage is managed by a SPARQL database, VitroLib provides a serialization of data by querying the database and retrieving the result also in JSON format.

To summarize, all the presented tools allow users to edit RDF data by means of input forms. However, none of these tools combines the form-based visualization with a graph-based representation of data. This feature is beneficial as it offers users a complete vision of the data structure. Most of the evaluated tools support the serialization of RDF data as a database storage mechanism. All the analyzed tools are distributed as web applications or software libraries that need to be installed and configured thus requiring users’ technical skills. Moreover, although the web-based approach enables users to work in a collaborative environment, it requires an active internet connection along the entire working process.

Finally, some of these tools support a template-driven approach, which allows to customize the features offered, including the data validation, on the basis of a configured template (like SHACL) at installation stage.

SHAPEness supports a SHACL-driven approach at the execution stage and this key feature, along with the others, will be widely described in Section 3.

3. SHAPEness features

The main goal in developing SHAPEness was to create a rich desktop application suitable for any context, domain, and use case that needs to browse, edit and validate RDF graphs, according to a set of input SHACL constraints (schema).

SHAPEness provides a rich user interface which combines i) a graph-based visualization, for helping users to analyze and better understand graphs’ structure and semantic; ii) a form-based visualization for easily editing graphs; iii) a tree-based visualization for exploring the graph by expanding parent nodes to view its more specific child nodes. The entire user interface is depicted in Figure 1 and described in Section 3.1.

The SHAPEness features are completely driven by a given SHACL schema (see Section 3.2), thus enabling users to easily structure RDF graphs according to the provided SHACL constraints (see Section 3.3).
The application assists users in properties compilation: a) by concealing the schema complexity; b) by helping avoid typos (see Section 3.4); and c) by validating property types and mandatory property (see Section 3.5). Section 3.6 describes how users can export the created RDF graphs.

3.1. User Interface

The SHAPEness user interface consists of five views: Graph View (Figure 1a), Palette View (Figure 1b), Outline View (Figure 1c), Properties View (Figure 1d), RDF/Turtle View (Figure 1e).

Users can customize the appearance of the application by resizing views and rearranging them using drag and drop within the main window.

The Graph View is a close representation of an RDF graph. The nodes represent resources and the edges represent relationships. Each node-edge-node connection represents a subject-predicate-object statement. A node is represented by an oval shape. An edge is represented by a labeled arrow which indicates the direction of the relation: from the subject to the object.

The view also includes: i) a toolbar which provides access to the common commands on the graph (e.g., zoom in, zoom out, change graph layout, add and remove nodes or relationships, etc.); ii) a filter panel which allows users to visualize or hide specific node types on the graph; iii) a context menu on nodes and edges which gives users a shortcut to frequently used commands.

The Palette View shows the shapes defined by the uploaded SHACL schema, through a list of colored circles. The purpose of this view is to allow users to create nodes of an RDF graph by simply dragging the shapes from this view to the Graph View, or alternatively by right-clicking on the shapes.

The Outline View represents the RDF graph as a tree-like structure where the nodes are grouped according to the shape types and the edges are represented by arrows in order to indicate the subject to the object of the relationships. This view is automatically updated according to the changes made in the Graph View.

The Properties View allows inspection and editing of node properties by providing several form fields (e.g., text fields, tables, dropdown lists, etc.). Each property is labeled by using its compact IRI expressed as prefix:suffix (e.g., foaf:name). The view is able to group the properties by mandatory, recommended and optional, on the basis of their cardinality (as defined by the uploaded SHACL schema).
The RDF/Turtle View provides the RDF/Turtle serialization of the graph. By default, this view is hidden. When activated, the application goes into a split-view mode and shows the Turtle serialization automatically generated according to the changes made in the Graph View or Properties View.

3.2. Using a SHACL schema to set up SHAPEness

SHAPEness implements a SHACL-driven approach for creating, editing, and validating RDF graphs. A typical user interaction workflow includes: a) selection of a SHACL schema (from URL or local file); b) creation of a new RDF graph compliant with the selected SHACL schema; or c) import of an existing RDF data graph serialized in RDF/Turtle format (from URL or local file). Finally, the application is able to dynamically build up the user interface and its underlying data model based on the SHACL shapes, in order to allow the user to work with graphs. These steps lead towards creating a SHAPEness project which will be saved, by default, to a dedicated folder inside the user workspace. SHAPEness creates two subfolders for each project folder. One for saving the SHACL schema (which drives the project) and all standard ontologies needed for the graph validation (e.g., foaf, vcard, skos, dcat, etc.). Another one for saving the graph automatically serialized in Turtle format. Saving the work as a project allows the user to open it in later stages in order to finalize or refine it, or even to share it in a team.

3.3. Browsing and Authoring RDF graphs

Three of the views, described in section 3.1, are devoted to providing a suite of operations which allow to explore and structure RDF data graphs: Palette View, Graph View, Outline View.

Users can create new nodes by double-clicking or right-clicking on the shape in the Palette View, as well as by dragging-and-dropping a shape from the Palette View to the Graph View.

The Graph View presents an overall picture of the entities and relationships via graph-based intuitive interface, and allows for considerable customization of the visual elements. SHAPEness assigns different colors to the shapes type listed in the Palette View. Nodes on the graph are labeled and color-coded according to their shape type. Users can customize these colors by means of the palette toolbar or the context menu on the nodes, at any time.

The graph is interactive, so nodes and edges can be moved around for a better inspection and selection. Every element of the graph, be it a node or an edge, when clicked is highlighted with a black border.

With a growing number of nodes and edges, the graph could become difficult to read and manage. To overcome this limit SHAPEness provides zoom commands and a filter panel which allows to hide specific nodes on the graph. Furthermore, the Outline View provides a free-text search which allows to quickly find nodes and edges on the graph.

3.4. Assisting properties compilation

SHAPEness empowers metadata authors and data curators to meet SHACL compliance requirements faster and more accurately by assisting the properties compilation.

The Properties View enables users to view and edit the properties of a selected node on the graph. In order to facilitate the acquiring of property values, SHAPEness gathers dynamically information about types, allowed values, and cardinality of the properties from the SHACL schema, and creates suitable form fields (e.g., text fields, dropdown lists, tables, etc.), as well as dedicated widgets to avoid typos. For example, as shown in Figure 2a, to enter geospatial information (points or polygons), an interactive Map dialog is provided. It enables users to draw a rectangle on a map or fill in geographic coordinates. Concerning date/time properties, the application provides a widget which allows to select a date/time from a graphical calendar (Figure 2b).
Furthermore, SHAPEness is able to gather information about the meaning of the properties from the SHACL schema or related ontologies, if they are available. A help button functionality is provided to assist users in understanding the meaning of IRI labels. For instance, in the case of “dct:isVersionOf” property for a dcat:Dataset entity users can get the term definition, i.e.: "A related resource of which the described resource is a version, edition, or adaptation.", as defined by the Dublin Core ontology (Figure 2c).

3.5. Validating RDF graphs

SHAPEness implements on-the-fly quality and consistency check of the graph, validating its content according to the SHACL constraints. Users receive notifications about constraint violations of invalid portions of the graph.

The notification mechanism includes alerts about two types of violations: errors and warnings. An error represents a critical problem, related to mandatory properties: it invalidates the graph and requires it to be necessarily fixed. A warning represents a not critical problem (an advice), related to recommended properties: it should be fixed in order to improve the quality of the graph.

Violations are notified in two views of the user interface (Figure 3). In the Graph View, an alert icon is visualized inside the oval shapes of those nodes that contain faulty properties. In the Properties View, an alert icon appears close to the property fields that fail the validation. In the upper side of the view an error message dialog reports the total number of detected violations and a brief description of them.

3.6. Exporting RDF graphs

SHAPEness automatically serializes the RDF graph into Turtle format and saves the file to a dedicated folder. Users may export the Turtle file to another local folder or push it to a Git repository. In this case, users can perform the push by inserting the Git repository URL, the branch name and the authentication credentials. This feature is useful for sharing RDF graphs within collaborative frameworks, or even for triggering a data ingestion pipeline within automated processes.

Furthermore, SHAPEness offers the possibility to export the RDF graph as a PNG image, this is particularly convenient for documentation purposes.

---

11 https://udfr.org/docs/onto/dct_isVersionOf.html
The design of the SHAPEness application was conducted considering the following requirements:

1) SHACL-driven: the application behavior should be dynamically driven by a set of SHACL constraints given as input.
2) User friendly interface: providing a graph-based interface for exploring RDF graphs, and a form-based interface for editing graphs properties.
3) Offline usability: enabling users to work without an internet connection.
4) Multi-Platform: running it on any operating system.
5) Easily Extensible: new features should be easily added to the application.
6) Freeware: users should not pay for its usage.

Figure 4 depicts the SHAPEness architecture conceived to fulfill the above requirements.

**SHAPEness is implemented as a standalone Java application based on Eclipse Rich Client Platform (RCP) that is an open source platform for building rich desktop applications (requirement 3 and 6). By rich it is meant that the platform provides a broad feature set and a highly interactive and usable visual design (requirement 2). RCP includes the OSGi framework which enables the deployment of native GUI applications to a variety of desktop operating systems.**

---

12 https://www.eclipse.org/
14 https://docs.osgi.org/specification/osgi.core/8.0.0/
systems (requirement 4). Developing on the OSGi framework makes the components very modular and loosely coupled, therefore, new functionality can be easily integrated (requirement 5).

The SHAPEness application includes an Engine component developed to fulfill the requirement 1 and to build the dynamic SHAPEness data model.

4.1. Eclipse Rich Client Platform (RCP)

SHAPEness is based on Eclipse Rich Client Platform (RCP) which provides a minimum set of plugins to build a rich desktop application. As any RCP applications, SHAPEness uses the dynamic plugin model and the user interface is built with a set of toolkits and extension points. The extension points mechanism of Eclipse permits to build applications in which components are loosely coupled, making it easy to add/remove/replace them. Third-party tools can be easily integrated in the application by registering them as extensions. Such a plugin-based approach overcomes the maintainability and lack of modularity issues that may arise with monolithic applications. In addition, it provides extreme development flexibility, as new functionalities can be implemented by adding new components (plugins or extension points) without having to rewire the entire application.

The SHAPEness user interface is based on the Workbench plugin which provides the overall structure of Eclipse and many common extension points used to customize the application. Several views are designed in order to present the underlying SHAPEness data model (see Section 4.2) and allow users to perform data input. The views and other visual parts are developed by using the SWT\(^1\)\(^\text{17}\) and JFace\(^1\)\(^\text{16}\) frameworks, that are adopted to build a form-based interface which provides several widgets (such as text fields, dropdown lists, tables, trees, wizard, dialog, etc.) and simplifies data editing.

Eclipse, by default, does not include any toolkit to create and manipulate graphs. As a result, SHAPEness uses the Graphical Editing Framework\(^1\)![](https://www.eclipse.org/gef/) (GEF) which bundles three components: Draw2d\(^1\)\(^\text{18}\), the GEF framework, and Zest\(^1\)![](https://www.eclipse.org/zest/).

In order to provide users with a preview of the graph serialized in RDF/Turtle format, SHAPEness harnesses XTurtle\(^1\)![](https://www.oracl.com/java/technologies/javase/desktop.html) plugin. Xturtle is based on Eclipse/Xtext\(^1\)![](https://www.eclipse.org/xtext/) and allows syntax highlighting, code completion, templates, syntax validation, internal linking to descriptions, resource preview, structure navigation and quick structure, folding options and multiple customization, including syntax illumination.

Finally, by integrating the JGit\(^1\)![](https://www.eclipse.org/jgit/) plugin, which is a pure Java implementation of the Git version control system, SHAPEness allows users to share their work with collaborators or with a wider audience, by pushing the serialized graph (i.e. Turtle file) to a Git repository.

4.2. SHAPEness Engine

SHAPEness Engine is responsible for loading the input SHACL file, building the domain model classes behind the application, and serializing/deserializing the model objects to RDF/Turtle.

4.2.1. Loading SHACL content

The Engine loads a set of SHACL shapes and constraints from a local file or a URL, and stores the content in memory, as a shapes graph, by using Apache Jena\(^1\)![](https://jena.apache.org/) Java library. The shapes graph contains two types of shapes. Node shapes which declare constraints on a node and Property shape which declares constraints on the attributes of a node through a path property.

4.2.2. Generating model classes

Typically, in a Java application, the application domain is modelled by means of JavaBean\(^1\)![](https://www.eclipse.org/jgit/) classes which are then used to populate model objects. The model classes are used not only to describe the concepts used in an application and their relationships, but also to define the behavior i.e. use case methods/functions that are appropriate to a specific domain. The JavaBeans can be created i) at design phase, i.e. classes are designed and developed at Java source code level; ii) at runtime phase, i.e. classes are created in a running program, without the need for recompilation and redeployment.

In order to implement the SHACL-driven approach, which requires the runtime creation of SHACL shapes, represented by JavaBeans, advanced Java programming techniques were adopted. As a

\(^{17}\) https://www.eclipse.org/swt/
\(^{16}\) https://wiki.eclipse.org/JFace
\(^{15}\) https://www.eclipse.org/gef/draw2d/index.php
\(^{19}\) https://www.eclipse.org/zest/
consequence, the Engine was developed by using the Java Reflection mechanism enhanced by Javassist library. Such library allows for manipulation of Java bytecode files, as well as runtime compilation of dynamically created classes, thus changing the runtime behavior of the application running in the Java virtual machine.

The Engine is able to create JavaBeans according to the Oracle JavaBeans Standard. Namely, each of them contains private fields that are accessible and modifiable by public methods (getters and setters), a no-arguments constructor, and implement the Serializable interface. JavaBean class names, field names, field types, as well as field constraints are dynamically defined by exploring the shapes graph recursively.

Figure 5 and Figure 6 show the pseudocode of the algorithm used by the Engine to generate the JavaBeans from a given SHACL content. The algorithm is divided in two parts. Figure 5 shows the graph exploration procedure which invokes the recursive algorithm able to create JavaBeans (Figure 6).

![Algorithm 1. Explore Shape Graph.](image)

**Algorithm 1. Explore Shape Graph.**

Input:
GRAPH ∈ A SHACL Shapes graph which contains a set of RDF Triples

Begin Explore Shape Graph Procedure
1. For Each RDF Triple in GRAPH do:
2. SUBJECT ∈ Subject Node of RDF Triple
3. PREDICATE ∈ Predicate of RDF Triple
4. OBJECT ∈ Object Node of RDF Triple
5. If PREDICATE equals "a" and OBJECT equals "ex:NodeShape" then:
   6. SHACL_to_JavaBeans (GRAPH, SUBJECT)
7. End
8. Done
End Explore Shape Graph Procedure

Fig. 5. The pseudocode of the algorithm used by the SHAPEexs Engine for exploring a SHACL shapes graph.

![Algorithm 2. SHACL to JavaBeans.](image)

**Algorithm 2. SHACL to JavaBeans.**

Input:
GRAPH ∈ A SHACL Shapes graph which contains a set of RDF Triples
NODE ∈ A node contained in GRAPH

Output:
JAVABeans ∈ A list of JavaBeans classes

Begin SHACL_to_JavaBeans Procedure
1. For Each RDF Triple in GRAPH with Subject equals to NODE do:
2. SUBJECT ∈ Subject Node of RDF Triple
3. PREDICATE ∈ Predicate of RDF Triple
4. OBJECT ∈ Object Node of RDF Triple
5. If PREDICATE equals "a" and OBJECT equals "ex:NodeShape" then:
6. If JAVABeans does not contain a JavaBean class named as SUBJECT then:
   7. JAVABean ∈ new JavaBean class named as SUBJECT value
   8. JAVABeans.add (JAVABean)
   9. End
10. Else
11. JAVABean ∈ existing JavaBean class
12. Done
13. End
14. End
15. End
16. JAVABean ∈ create a new field named as OBJECT value
17. SHACL_to_JavaBeans (GRAPH, OBJECT)
18. End
19. End
20. If JAVABeans contains a field named as SUBJECT value then:
21. JAVABean ∈ create a new field named as OBJECT value
22. FIELD_NAME ∈ OBJECT value
23. End
24. If PREDICATE equals "a" and OBJECT equals "ex:NodeShape" then:
25. FIELD_TYPE ∈ OBJECT value
26. If OBJECT is a "ex:NodeShape" in GRAPH then:
27. SHACL_to_JavaBeans (GRAPH, OBJECT)
28. End
29. End
30. If PREDICATE equals "a" and OBJECT equals "ex:NodeShape" then:
31. FIELD_TYPE ∈ create object
32. End
33. If PREDICATE equals "a" and OBJECT equals "ex:NodeShape" then:
34. FIELD_TYPE ∈ List
35. End
36. If PREDICATE equals "a" and OBJECT equals "ex:NodeShape" then:
37. FIELD_TYPE ∈ List
38. End
39. If PREDICATE equals "a" and OBJECT equals "ex:NodeShape" then:
40. FIELD_TYPE ∈ List
41. End
42. If PREDICATE equals "a" and OBJECT equals "ex:NodeShape" then:
43. FIELD_TYPE ∈ List
44. End
45. End
46. End
47. JAVABeans ∈ add JAVABean
48. Done
End SHACL_to_JavaBeans Procedure

Fig. 6. The pseudocode of the recursive algorithm used by the Algorithm 1 for generating Java classes from SHACL shapes.

Figure 7 provides an example of how this algorithm generates a JavaBean class, named Person, by using a set of constraints defined for a NodeShape named ex:PersonShape, shown in Figure 8. As shown in the example, the Engine uses the JavaBean Validation with custom annotations, placed on fields of JavaBean classes, in order to specify the SHACL properties constraints (e.g., minimum and maximum cardinality, allowed values, etc.). Such annotations allow to perform the validation of the constraints through the various application components.

25 https://www.oracle.com/technical-resources/articles/java/javareflection.html
26 https://www.javassist.org/
27 https://www.oracle.com/java/technologies/javase/javabeans-spec.html
28 https://docs.oracle.com/javaee/7/tutorial/bean-validation001.htm
Moreover, each JavaBean is defined as a subclass of SHAPEness Node class which represents a generic node entity of the SHAPEness data model. Inheritance allows us to take advantage of polymorphism because different subclasses of Node have the same capabilities (i.e., provide the same methods), even though the subclasses implement their methods differently. This allows the Engine to use an instance of a subclass wherever an instance of Node is needed and therefore to manipulate, at runtime, objects differently depending on their class type.

4.2.3. Serializing/deserializing the data model to RDF/Turtle

Once the model classes are built on the basis of a given SHACL file, users are able to create new instances of these classes by means of the graph-based user interface.

Instances of any domain model class are considered by SHAPEness as instances of the Node super-class, which represents a generic node of an RDF graph. Each node (Java object) has its own state which basically represents a snapshot of the object at a given time, i.e. properties/attributes and behaviors (actions they can or have performed on them).

The Engine is responsible to save the state of all nodes created on the graph and transform these objects into a stream of bytes objects by means of an RDF resource serialization process.

The Engine is also able to perform the reverse process. Namely, it is able to restore the state of objects through a deserialization process in which the byte stream is transformed back into Java objects (i.e. nodes of an RDF graph).

The Engine uses RDF/Turtle as the serialization and deserialization format and leverages the Apache Jena API to implement both mechanisms.

5. Application of SHAPEness in EPOS

The motivation for developing SHAPEness was triggered by the concrete requirements of a challenging Research Infrastructure (RI) for solid Earth sciences: the European Plate Observing System 29 (EPOS). EPOS is a prominent example of Information-Powered Collaborations (IPC) — the approach devised and adopted to enable cross-disciplinary knowledge pooling in such complex en-

---

environments is extensively described in previous work [19, 20]. A major challenge in EPOS is the integration of multi-disciplinary, multi-organizational, distributed resources into a single overarching RI — the EPOS Integrated Core Services (ICS). ICS provide users with a harmonized view of diverse community assets, i.e. Thematic Core Services (TCS), that contribute domain-specific resources such as data and metadata products, software and services. Interoperability is achieved via canonical representations of such resources. Being EPOS committed to provide the integrated resources according to the FAIR principles, the metadata challenge is of primary importance and the provision, management and production of metadata descriptions plays a fundamental role in the EPOS Software Development Life Cycle [1].

To tackle the inherent socio-technical challenges in EPOS a methodology was applied to separate concerns and build a common information space that underpins the IPC. The CRP methodology encompasses three dimensions to be addressed independently: “Conceptual definition (C), Representation (R), Population (P). C develops the ways of thinking, introducing terminology and meaning into the knowledge space, R develops detail and how to organise concepts, P deals with the gathering of instances of concepts in order to meet a community’s requirements” [19].

A rich and flexible data model was designed for the representation of such concepts and their relationships, namely EPOS-DCAT-AP [30], which is a DCAT [31] application profile available in RDF format and structured as a SHACL Shapes graph, thus supporting validation and consistency checks [21]. To populate such a representation an ingestion process (pipeline) was devised. SHAPEness elaborates on those results and offers a generic and scalable solution for the implementation of knowledge graph population pipelines.

In EPOS, major requirements focused on usability by non-technical domain experts and applicability to the diverse and heterogeneous communities composing the RI. Although each domain might have different policies for data and metadata management, they eventually exchange the information required by the EPOS RI by populating the EPOS-DCAT-AP model. In the first phases of the project the population and curation were mostly human-based processes, e.g. by manually editing and validating RDF/Turtle files according to the EPOS-DCAT-AP model.

The SHAPEness editor provides a significant added value and its features, described in Section 3, fulfill EPOS’ requirements. It enables automation and enhancements in the creation and population of quality checked EPOS-DCAT-AP graphs by distributed and heterogeneous communities.

Figure 9 illustrates the processes that compose the EPOS Metadata Ingestion Pipeline. Community Metadata Authors have the responsibility to provide ICS with the metadata from their TCS according to ICS’ requirements i.e. by populating EPOS-DCAT-AP RDF graphs. The creation of an EPOS-DCAT-AP RDF graph is followed by a validation of its SHACL constraints (e.g. structural and syntactical) and a visual inspection and curation of its content. In the next phase the RDF graph is serialized in a file and deposited onto a dedicated TCS shared space (e.g. GitLab). This action triggers the automated ingestion of the file to the ICS. ICS receives inputs from multiple TCS pipelines, which represent the ten EPOS communities, and integrates the corresponding EPOS-DCAT-AP RDF representation in a consistent consolidated knowledge graph. Once the integrated metadata is validated it requires the final approval of an EPOS Metadata Curator in order to be applied in the production environment.

As illustrated in Figure 9 SHAPEness plays a crucial role in several steps of the ingestion pipeline: a) it provides TCS Metadata Authors with a powerful environment where they can create new EPOS-DCAT-AP RDF files and validate them; b) it offers an interactive tool for browsing and assessing the quality of the RDF graph content; c) it integrates with shared repositories such as GitLab; and d) it can be adopted to support content validation and approval by the EPOS Metadata Curators in the ICS.

In the next section the approach adopted to assess the usability of SHAPEness in the EPOS community is described and the evaluation results are presented.

20 https://github.com/epos-eu/EPOS-DCAT-AP
6. User Testing

A testing plan was performed for the first release of the application in the framework of the EPOS community, described in the previous section. The testing included four main steps: a) definition of the objectives; b) definition of the methodology; c) submission of testing questionnaire; d) collection of results. These steps, described in the following subsections, compose a structured testing plan that was executed partially in the first release of the software presented in this work. This first round of testing provided inputs for further software developments and, by making the application more popular in the EPOS community, paved the way for additional tests with a wider user-base.

6.1. Testing Objectives

In order to draw clear boundaries and define the purpose of the testing, the first step was to define what aspects were going to be addressed in the testing.

It was decided to perform a user testing, excluding other types of technical testing aiming at validating the quality or structure of the architecture and code, as the structural or behavioral testing [2], both already performed to some extent as routine procedures in the software development practice (Continuous Integration through GitLab\textsuperscript{32}); the aim was therefore to collect and analyze the following elements related to the user experience:

- Validation of the application in terms of a) functionalities, to verify that the software responds to user expectations, and b) usability, to check that the software usage is easy and straightforward.
- Collection of user inputs to improve the quality and utility of the software; this includes collection of new requirements, bugs notification, improvement of existing functionalities.
- Collection of user profile related requirements: different types of users may have different or even divergent expectations with respect to the SHAPE\textit{ness} application; knowing what requirements are raised by what user profiles, allows the developer to prioritize implementation of new features on the basis of available resources, time constraints and target user groups.

\textsuperscript{32} https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/
6.2. Testing methodology

A testing methodology was designed and included four main elements: a) definition of user groups; b) definition of a questionnaire; c) questionnaire submission; d) collection and analysis of results.

Potential users were categorized in three different user groups:

1) Average skilled computer users (e.g. use office, excel, and other common apps), with very low knowledge of the metadata concept. These users are particularly interesting for their feedback about the usability of the user interface and the intuitiveness of the flow of actions.

2) Metadata practitioners with a basic technical knowledge of EPOS-DCAT-AP. These users’ focus is on the usability of functionalities to create RDF graphs (e.g. compilation of forms, export functionalities, etc.). Their view might be influenced by the comparison with existing metadata tools.

3) Metadata practitioners with good knowledge of EPOS-DCAT-AP. These users’ feedback is particularly important to evaluate how SHAPEness supports the semantic features of the EPOS-DCAT-AP model, for instance representation of entities, relationships, correctness of the serialization and vocabulary information.

Each user group was expected to interact with the SHAPEness in a different way based on their skills and background. The structure of the questionnaire included a general introduction, questions about the installation procedure, use cases execution and feedback collection; in particular, the following questions were included:

- User profile personal information (e.g. background, experience);
- User profile technical information (e.g. Browser, Operating System);
- Questions about software installation;
- Questions about use case execution;
- Questions related to bugs;
- Questions related to functionalities;
- Questions related to usability;
- Generic questions about satisfaction index.

The testing program was advertised during EPOS meetings. Six out of fifteen beta testers who had been selected accepted to respond to the Google web forms questionnaire. Finally, the results were collected by means of spreadsheets and an analysis was conducted as described in the results section.

6.3. Results

6.3.1. User profiling

Six users from five different European countries answered the questionnaire. They all belong to research institutions. 33% of the users have a technical background (computer science), 13% have any hybrid technical/scientific background and 50% come from the scientific community (i.e.: seismology and remote sensing). 100% of users are familiar with EPOS-DCAT-AP. All users have authorized the processing of personal data in accordance with EU Regulation 2016/679. We conclude that 100% of users belong to user category #3 (metadata practitioners with good knowledge of EPOS-DCAT-AP).

6.3.2. Software Usage

The software was properly downloaded by all users, installation was done by 50% of users on Windows, 33% on Unix/Linux, 16,7% on iOS. 100% of users found the software easy to install and easy to open. After the installation, users were required to use the software being guided by two use cases, each of which split in several tasks: a) Creation of an RDF/Turtle file from scratch, including 6 tasks; b) Creation of a project from an existing RDF/Turtle file, including 7 tasks. Also, final questions were posed to collect general feedback.

For initializing the application, users were provided with a sample SHACL file.

Results of the execution of the first use case, “Creation of an RDF/Turtle file from scratch”, are shown in Table 1. The use case execution was therefore considered easy. Comments by users also provided additional information, in particular: a) small bugs were reported; b) no criticalities where emphasized with respect to the current functionalities, no new functionalities were required, although improvement of current functionalities is sometime suggested, as in the case of licenses metadata field (Task 3); c) usability of a few layout parts or actionable elements within the GUI needs to be improved, as in the case of the spatial coverage.

---

33 https://www.google.com/forms/about/
Results of the execution of the second use case, "Creation of a project from an existing RDF/Turtle file", are shown in Table 2. The use case execution was considered moderately easy. In this case, comments in each task were valuable and provided the following information: a) small bugs were reported, for instance deletion not saved upon closing the application; b) a few criticalities were emphasized with respect to the current functionalities, mostly related to the graph visualization; (Tasks 2-5); c) as a consequence, the application needs to improve the usability of the graph, and suggestions provided by users are very useful in this perspective.

Questions were included to have a general evaluation of the satisfaction about software. 50\% of the testers never used a similar application before. And, as illustrated in Figure 10, the overall evaluation is positive, emphasizing that the application needs to be improved to fully match users’ expectations.

Final questions focused on features, unexpected behavior or bugs, and additional comments/suggestions. Interestingly, the most appreciated features are those that the authors consider as the innovative added value of the software, i.e. a) graph automatic validation, b) easy to use by unexperienced person, c) configurable on the basis of runtime loadable SHACL files, d) easy creation of nodes and relationship. The least interesting features are those related to the coloring of nodes (4 out of 6 consider these least important). Few bugs were also reported, related to lag in the GUI response, and to different behavior on OSX operating systems. Brand new features were not really required, but few features for improving the graph navigation were suggested (e.g. additional dynamic layout for the bubble graph).

Table 1

<table>
<thead>
<tr>
<th>Task</th>
<th>Successful Execution</th>
<th>Avg Difficulty Index (Easy=1, Hard=5)</th>
<th>Relevant comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1: Creation of a new project</td>
<td>100%</td>
<td>1</td>
<td>Colors layout in dialog box; missing references to project in GUI layout.</td>
</tr>
<tr>
<td>T2: Creation of a dataset node with properties</td>
<td>100%</td>
<td>1</td>
<td>Dataset identifier input and update unclear; The dataset node comes with three errors</td>
</tr>
<tr>
<td>T3: Creation of a distribution node with properties and link it to dataset</td>
<td>100%</td>
<td>1.83</td>
<td>Link to dataset might be more easy; license field needs to be wider.</td>
</tr>
<tr>
<td>T4: Specify dataset spatial coverage</td>
<td>83%</td>
<td>2.33</td>
<td>Not intuitive; coordinate system not specified; not clear why and ID is needed</td>
</tr>
<tr>
<td>T5: Preview RDF/ttl</td>
<td>100%</td>
<td>1.17</td>
<td>The icon for preview of the turtle/rdf file is a little misleading</td>
</tr>
<tr>
<td>T6: Export RDF/ttl file</td>
<td>100%</td>
<td>1</td>
<td>It seems that the project is not saved if not exported.</td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td><strong>1.39</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Results of the execution of the second use case, "Creation of a project from an existing RDF/Turtle file", are shown in Table 2. The use case execution was considered moderately easy. In this case, comments in each task were valuable and provided the following information: a) small bugs were reported, for instance deletion not saved upon closing the application; b) a few criticalities were emphasized with respect to the current functionalities, mostly related to the graph visualization; (Tasks 2-5); c) as a consequence, the application needs to improve the usability of the graph, and suggestions provided by users are very useful in this perspective.

Questions were included to have a general evaluation of the satisfaction about software. 50\% of the testers never used a similar application before. And, as illustrated in Figure 10, the overall evaluation is positive, emphasizing that the application needs to be improved to fully match users’ expectations.

Final questions focused on features, unexpected behavior or bugs, and additional comments/suggestions. Interestingly, the most appreciated features are those that the authors consider as the innovative added value of the software, i.e. a) graph automatic validation, b) easy to use by unexperienced person, c) configurable on the basis of runtime loadable SHACL files, d) easy creation of nodes and relationship. The least interesting features are those related to the coloring of nodes (4 out of 6 consider these least important). Few bugs were also reported, related to lag in the GUI response, and to different behavior on OSX operating systems. Brand new features were not really required, but few features for improving the graph navigation were suggested (e.g. additional dynamic layout for the bubble graph).

![The SHAPEness usability evaluation questions](image_url)
In conclusion, on the basis of the questionnaire’s results, the SHAPEness application demonstrated to be easy to use with the proposed use cases, to have a good overall usability, and to satisfy the expectations of the expert users. It still needs improvements either in terms of bug fixing and in terms of new features related to the graph navigation.

In addition, SHAPEness was presented in the framework of ENVRI-FAIR European cluster project [17] training activities and encountered positive feedback.

7. Conclusions and future work

In this paper a Java desktop application, SHAPEness was presented. It supports metadata authors, curators, as well as non-expert users in order to meet SHACL compliance requirements of RDF data faster and more accurately.

SHAPEness leverages Eclipse Rich Client Platform (RCP), combined with additional frameworks, to build a rich user interface which is dynamically configured by taking as input a set of SHACL constraints that describe the expert knowledge of a domain.

In particular, based on the given SHACL content, SHAPEness is able to set up:

- a graph-based user interface which allows users to easily explore and structure RDF data as graphs according to the SHACL shapes;
- a form-based user interface which assists users in filling in nodes properties by providing suitable widgets and useful suggestions for a full compliance with SHACL constraints;
- a tree-based user interface which allows users to browse nodes and edges with by using a common structure;

Table 2

Results of the execution of “Creation of a project from an existing RDF/Turtle file” use case.

<table>
<thead>
<tr>
<th>Task</th>
<th>Successful Execution</th>
<th>Avg Difficulty Index (Easy=1, Hard=5)</th>
<th>Relevant comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1: File import</td>
<td>100%</td>
<td>1.50</td>
<td>Need a waiting message to inform user the process is running; need to restart the import process upon input of wrong URL.</td>
</tr>
<tr>
<td>T2: Explore nodes on the graph and change graph layout.</td>
<td>83%</td>
<td>n/a</td>
<td>Need more clear command to open the graph; the graph didn't show up</td>
</tr>
<tr>
<td>T3: Check the nodes with empty mandatory properties and fill them in.</td>
<td>100%</td>
<td>1.67</td>
<td>A window/tab with a summary of error/warnings may be useful; person node not responding, program stuck; It was not clear at first sight how to insert the postal code.</td>
</tr>
<tr>
<td>T4: Look for the node &quot;Concept <a href="">epos:SeismicStation</a>&quot; and delete it.</td>
<td>83%</td>
<td>2.00</td>
<td>Node deletion should be feasible from the Outline area too; graph layout could have an 'auto-focus'; finding the node via the graph could be not easy when the graph starts to be complex; the Outline area is very helpful in this context.</td>
</tr>
<tr>
<td>T5: Look for the relationship from WebService to ApiDocumentation and delete it.</td>
<td>100%</td>
<td>2.67</td>
<td>A graphical way to delete relation should be available too; Not immediate to find the relationship through the graph; Outline area is better</td>
</tr>
<tr>
<td>T6: Change the colour for all ContactPoint nodes.</td>
<td>100%</td>
<td>1.00</td>
<td>Colors of object should not be randomly assigned every time, but should be the same</td>
</tr>
<tr>
<td>T7: Delete all nodes and close the application</td>
<td>100%</td>
<td>1.16</td>
<td>the deletion was not saved</td>
</tr>
</tbody>
</table>

Mean 1.67
a validation process of RDF graphs which includes visual alerts about detected SHACL violations and suggestions on how to fix them. SHAPEness has been developed within the European Plate Observing System (EPOS) framework, where in order to exchange knowledge between the diverse EPOS Thematic Core Services (TCS) and the Integrated Core Services (ICS), an extension of DCAT Application Profile, called EPOS-DCAT-AP, has been conceived.

SHAPEness was formally tested by different users of the EPOS community who have evaluated its features, provided feedback about few bugs, partially fixed in the latest version, and who generally appreciated its added value as a tool for supporting the creation and validation of RDF graphs within the EPOS ingestion pipeline. An informal feedback provided by the ENVRI-FAIR community confirmed that the tool is fit for purpose and innovative.

Future plans include tackling open issues, addressing the inputs provided by current users and updating the tool accordingly.

The following additional features are planned for future release:

- serialization and deserialization of RDF graphs to other standard formats (e.g., RDF/XML, Notation-3);
- import and export of graphs from/to an RDF/Triple data stores in order to enable collaborative knowledge management;
- automated updates of the application by exploiting the Eclipse platform;

The tool is now distributed as a binary version because the release of an open source project requires additional efforts in terms of documentation, code quality check and license selection. These activities are planned and soon SHAPEness will be released as open-source.

Thanks to its self-adaptability, i.e., its capability to dynamically modify its behavior, based on a set of SHACL shapes given as input, SHAPEness proves to be suitable for any context, domain, and use case which uses SHACL language for describing and constraining the contents of RDF data.

Because of its ease of use, SHAPEness is a promising metadata management tool for metadata curators and practitioners, but specifically addressed to scientific experts with a limited semantic technology background that have to deal with metadata. This happens for instance when non-experts are required to take the responsibility for the creation, maintenance and update of quality and rich metadata produced in the framework of initiatives that aim at addressing the FAIR guiding principles for data stewardship.

Software availability

SHAPEness binaries are freely distributed under the GNU General Public License GPL-3. Download, setup instructions and user manual are available at GitHub site24.

Acknowledgments

This work has been supported by the EPOS Research Infrastructure through the contribution of the Italian Ministry of University and Research (MUR) and ENVRI-FAIR project (Horizon 2020) – Grant agreement No. 824068.

References


24 https://epos-eu.github.io/SHAPEness-Metadata-Editor/


